N° d'ordre: 000

Thèse

présentée

devant l'Université Joseph Fourier

pour obtenir

le grade de Docteur de l'Université Joseph Fourier
Mention Informatique

par

Aldric Degorre

Équipe d'accueil : Hybrid and Timed Systems
École Doctorale : MSTII
Composante universitaire : Vérimag

Titre de la thèse :

On Some Quantitative Aspects of Formal Languages

À soutenir le 21 octobre 2009 devant la commission d'examen.
Composition du jury :

Président

Rapporteurs
Dominique Perrin
Jean-François Raskin
Examinateurs
Eugene Asarin
Paul Gastin
Claude Jard
Yassine Lakhnech
Oded Maler (Directeur de thèse)

Contents

Chapter 1

Introduction

Formal languages are sets of sequences over a discrete set of symbols called *alphabet*. They can be specified by various means such formulae in some logic that express order relations, by regular expression or by discrete automata of various sorts that recognize these languages by accepting sequences that induce runs leading to accepting states (or accepting cycles, in case of infinite sequences) in the automaton. It is fair to say that the theory of formal languages is very *qualitative* in the following respects:

- Its objects are sequences, that is, mapping from a discrete/qualitative time domain to an alphabet which is typically discrete and non-metric;
- Acceptance criteria for sequences depend on whether or not the runs of an automaton visit accepting states or cycles. Such criteria can distinguish only between 0, 1 and ∞ visits;
- The theory does not pay much attention to quantitative comparison between languages in terms of size but rather on qualitative measures such as inclusion.

This thesis is a contribution to the study of some of these neglected issues using three classes of problems:

- *Timed Behaviors and Dynamic Scheduling*: One the most extensively studied quantitative extensions to formal languages is the addition of metric "real" time, and considering timed behaviors such as Boolean signals or time-event sequences [6]. The study of timed languages is rich both in theory (timed automata, real-time temporal logic, timed regular expressions) and in applications, most notably the modeling of dynamic scheduling problems. Chapter 2 is devoted to some fundamental results concerning a new scheduling model. In this model, an infinite stream of *structured* jobs (each job being a partially-ordered set of tasks) is to be scheduled on an execution platform consisting of a finite number of machines of various types. The set of admissible request streams is modeled as a timed ω-language over the finite alphabet of job types. On this model we prove some fundamental results such as the non-existence of a schedule of bounded latency

when the different job types are not compatible and do not "pipeline" easily, and present a scheduling *strategy* which can guarantee bounded backlog despite the uncertainty concerning the identity of future jobs.

- *Quantitative Acceptance Criteria*: In many contexts we would like to evaluate system behavior not only by qualitative criteria such as violation of properties, but also by quantitative criteria that reflect costs associated with transitions or states. In Chapter 3 we study some theoretical aspects of acceptance conditions based on multi-dimensional *mean-payoff*. We study the expressive power and topological complexity of such criteria and define a class of mean-payoff languages which is closed under Boolean operations and whose emptiness problem is decidable.

- *Volume and Entropy of Timed Languages*: In this major part of the thesis we combine two quantitative aspects of formal languages. We work in the context of timed languages and define the measures of volume and entropy for these language, measures that characterize the size of the language and provide for a quantitative comparison between timed language, for a example to assess the degree of over approximation of a system by its abstract model. The extension of the concepts of size and entropy to timed languages is particularly challenging due to the non countable nature of timed behaviors and timed automata. After giving definitions of these measures we develop several methods for computing and approximating them. This work opens new research directions concerning the information theory of Boolean signals and show interesting relations to algorithmic complexity, functional analysis and symbolic dynamics.

Chapter 2

Scheduling Streams of Structured Jobs

2.1 Introduction

Scheduling, in its most general sense, is the act of deciding when (and where) to execute tasks. However this definition is vague enough for a large number of researchers to claim being working in this field. The problem of efficient allocation of reusable resources over time, is indeed a universal problem, appearing almost everywhere, ranging from the allocation of machines in a factory [46, 14, 36], allocation of processor time slots in a real-time system [42, 19], allocating communication channels in a network [31], or allocation of vehicles for transportation tasks [15]. Unfortunately, the study of scheduling problems is distributed among many academic communities and application domains, each focusing on certain aspects of the problem.

The aspects in which scheduling models may differ range among the following:

- First in the way *resources* are considered. A resource is an entity with its own timeline, which can be allocated to tasks. Usually resources are machines, network links, computer parts, or even people. Resources may be reusable (like a processor) or not (like energy), may exist in several interchangeable *homogeneous instances* of a same type (like the cores of a multicore CPU) or be all different, *heterogeneous*.

- The *tasks*, the units of work to schedule, can exhibit some varying features. Tasks may have a predifined duration, and a set of resources they need to use. The exact set of tasks may be known before the system under observation is executed, or may not. Some *uncertainty* might be allowed in the duration of those tasks, or maybe in the time when tasks are *issued*, or even, it might not be known if some tasks will be issued at all. The task set can be finite or infinite. If it is infinite, instance of tasks may have either periodical or sporadical arrival patterns.

- General rules may also differ. Tasks may have a *precedence* relation (*dependencies*), i.e. a task might have to wait for all (or, in some models, at least one) of

its predecessors before starting. Resources can also be *preemptible*, meaning that a task can be interrupted before it is finished. In this case, the resource becomes availabe for another task of higher priority.

However, in the vast scheduling literature, one can, very roughly, identify two generic types of problems. In the first type, the work to be scheduled admits a *structure* which includes precedence constraints between tasks, but the problems are, more often than not, *static*: the work to be executed is *known in advance* and is typically finite. Examples of this type of problems are the job-shop problem motivated by manufacturing (linear precedence constraints, heterogeneous resources) [36, 35] or the task-graph scheduling problem, motivated parallel execution of programs (partially-ordered tasks, homogeneous resources) [29]. Some recurrent aspects of scheduling are exhibited in program loop parallelization [27] but the nature of uncertainty there is different and rather limited.

On the other hand, in problems related to real-time systems [20] or in queuing theory [34], one is concerned with *infinite streams* of tasks which arrive either periodically or sporadically (or in a combination of both), satisfying some constraints on task arrival patterns. In many of these "dynamical" problems, the structural dimension of the problem is rather weak, and each request consists of a monolithic amount of work. A notable exception is the domain of adversarial queuing theory [16] where some structure and uncertainty are combined.

In this chapter we propose a model which combines the *dynamic* aspect associated with *request streams* whose exact content is not known in advance, with the *structural* aspects expressed by task dependencies. We define a scheduling problem where the demand for work is expressed as a *stream* of requests, each being a structured *job* taken from a finite set of types, hence such a stream can be viewed as a timed word over the the alphabet of job types. Each job type defines a finite partially-ordered set of tasks, each associated with a resource type and a duration. Such a stream is to be scheduled on an *execution platform* consisting of a finite number of resources (machines). A schedule is valid relative to a request stream if it satisfies *both* the precedence constraints imposed by the structure of the jobs and the resource constraints imposed by the number of resources available in the platform (and, of course, it does not execute jobs before they are requested).

The quality of a specific schedule is evaluated according to two types of measures, one associated with the evolution of the *backlog* over time, that is, the difference between the amount of work requested and the amount of work supplied, and the *latency*, the temporal distance between the arrival of a job instance and the termination of its execution. To model the uncertain external environment we use the concept of a *request generator*, a set of request streams satisfying some inter-arrival timing constraints. Such constraints can be expressed, for example, using timed automata [3], real-time logics [4] or timed regular expressions [6]. We restrict the discussion to *admissible* request streams that do not demand more work over time than the platform can offer. A *scheduling policy* (strategy) should produce a schedule for each admissible request stream, subject

to *causality* constraints: the decision of the scheduler at a given moment can only be based on the *prefix* of the request stream it has seen so far.

After defining all these notions we prove two major fundamental results:

- Positive: we develop a scheduling policy which produces a bounded backlog schedule for any admissible request stream. Note that due to the precedence constraints between the tasks in the jobs, request stream admissibility does not, a priori, guarantee the existence of such a schedule. In fact, we show that a naive "oldest first" policy can accumulate an *unbounded backlog* for certain request streams. Our policy achieves this goal by making decisions that provide for *pipelined* execution whenever possible.

- Negative: there are admissible request streams for which no bounded-latency schedule (and hence no bounded-latency policy) exists.

The rest of the chapter is organized as follows: in Sect. 2.2 we define our scheduling framework, in Sect. 2.3 we prove a negative result concerning the impossibility of bounded latency schedules. In Sect. 2.4 we extend the framework to include scheduling policies and in Sect. 2.5 we develop a scheduling strategy that guarantees bounded backlog. We conclude with a discussion of past and future work.

These results were first published in [28].

2.2 The Recurrent Scheduling Problem

2.2.1 General Definitions

We use timed words and timed languages to specify streams of requests for work. Intuitively, a timed word such as $\tilde{u} = 3a_1 2a_2 a_3 6$ consists of a passage of time of duration 3, followed by the event a_1, followed by a time duration 2, followed by the two events a_2 and a_3 and then a time duration of 6. We present some basic definitions and notations (see more formal details in [6]).

- A word over an event alphabet Σ is either ϵ, the empty word, or ua where u is a word and $a \in \Sigma$. An ω-word is an infinite sequence $(a_i)_{i\in\mathbb{N}} \in \Sigma^\omega$.

- A timed word over Σ is a word over $\Sigma \cup \mathbb{R}_+$. The duration of a timed word u, denoted by $|u|$ is the sum of its elements that are taken from $\mathbb{R}_+$, for example $|\tilde{u}| = 11$. A timed ω-word is an infinite sequence $(a_i)_{i\in\mathbb{N}} \in (\Sigma \cup \mathbb{R}_+)^\omega$ such that its duration diverges.

- The concatenation of a word u and a word (or ω-word) v is denoted by uv.

- A word u is a prefix of v iff there exists w such that $v = uw$, which we denote $u \sqsubseteq v$. We say that u is a proper prefix of v, denoted by $u \sqsubset v$, if $u \neq v$.

- A word (or an ω-word) u is a suffix of v iff there exists w such that $v = wu$.

For a timed (ω-)word u over Σ

- By $u(a, i)$ we denote the time of the i-th occurrence of event $a \in \Sigma$ in the timed word u. Formally $u(a, i) = t$ if $u = vaw$ such that $|v| = t$ and v contains $i - 1$ occurrences of a. We let $u(a, i) = \infty$ when a occurs less than i times in u.

- The timed word $u_{[0,t]}$ is the longest prefix of u with duration t. Formally $u_{[0,t]} = t_0 a_0 t_1 a_1 ... t_i$ such that $\sum_{0 \leq k \leq i} t_k = t$ and there exists no discrete event a such that $t_0 a_0 t_1 a_1 ... t_i a$ is a prefix of w. For example, $\tilde{u}_{[0,4]} = 3a_1 1$ and $\tilde{u}_{[0,5]} = 3a_1 2a_2 a_3 0$.

Sets of timed (ω-)words over Σ are called timed (ω-)language. We denote the sets of such languages by $\mathcal{T}(\Sigma)$ and $\mathcal{T}_\omega(\Sigma)$, respectively.

2.2.2 Execution Platform, Jobs and Tasks

The execution platform determines our capacity to process work.

Definition 1 (Execution Platform)**.** *An execution platform over a finite set $M = \{m_1, \ldots, m_n\}$ of resource (machine) types is a function $R : M \to \mathbb{N}$.*

Example: $\{m_1 \mapsto 2, m_2 \mapsto 4, m_3 \mapsto 1\}$ is an execution platform with three resource types m_1, m_2, m_3 having 2 instances of m_1, 4 instances of m_2, and 1 instance of m_3.[1]

The *task* is the atomic unit of work, specified by the resource type it consumes and by its duration. The job is a unit of a larger granularity, consisting of tasks related by precedence constraints. Each job is an instantiation of a job type.

Definition 2 (Job Type)**.** *A job type over a set M of resources is a tuple $J = \langle T, \prec, \mu, d \rangle$ such that $\prec \subseteq T \times T$ and $\langle T, \prec \rangle$ is a finite directed acyclic graph whose nodes are labelled by 2 functions: $\mu : T \to M$, which associates a task to the resource type it consumes, and $d : T \to \mathbb{R}_+ - \{0\}$ specifying task duration.*

As an example consider a job type where $T = \{a_1, a_2, a_3\}$, $\prec = \{(a_1 \prec a_3), (a_2 \prec a_3)\}$ $\mu = \{a_1 \mapsto m_1, a_2 \mapsto m_2, a_3 \mapsto m_3\}$, $d = \{a_1 \mapsto 3, a_2 \mapsto 2, a_3 \mapsto 1\}\rangle$, where a_1 needs resource m_1 for 3 time units, a_2 uses resource m_2 for 2 time units while a_3 consumes m_3 for 1 time unit. Task a_3 cannot start before both a_1 and a_2 terminate.

For a set $\mathcal{J} = \{\langle T_1, \prec_1, \mu_1, d_1 \rangle, ..., \langle T_n, \prec_n, \mu_n, d_n \rangle\}$ of job types, we let $T_\mathcal{J}, \prec_\mathcal{J}, \mu_\mathcal{J}$ and $d_\mathcal{J}$ denote, respectively, the (disjoint) union of T_i, $\prec_i$, μ_i and d_i, for $i = 1..n$. We call elements of $T_\mathcal{J}$ *task types*. When $\mathcal{J}$ is clear from the context we use notations T, $\prec$, μ and d.

Definition 3 (Initial Tasks, Rank)**.** *An initial task a is an element of T such that there exists no $a' \in T$ with $a' \prec a$. The rank of task a is the number of edges of the longest path $a_0 \prec a_1 \prec \cdots \prec a$ such that a_0 is initial. Initial tasks have rank 0.*

[1] We will use the notation R_m for $R(m)$ and R when we want to treat the whole platform capacity as vector and make component-wise arithmetical operations. The same will hold for sets of functions indexed by the elements of M.

2.2.3 The Demand

The sequence of jobs and tasks that should be executed on the platform is determined by a request stream.

Definition 4 (Request Streams and Generators)**.** *A request stream over a set $\mathcal{J}$ of job types is a timed ω-word over $\mathcal{J}$. A request generator is a timed ω-language over $\mathcal{J}$.*

Each request stream presents a demand for work over time which should not exceed the platform capacity, otherwise the latter will be saturated.

Definition 5 (Work Requested by Jobs and Streams)**.** *With each resource type m we define a function $W_m : \mathcal{J} \to \mathbb{R}_+$ so that $W_m(J)$ indicates the total amount of work on m demanded by job J, $W_m(J) = \sum_{\{a \in T_J : \mu(a)=m\}} d(a)$. We lift this function to request stream prefixes by letting $W(\epsilon) = 0$, $W(ut) = W(u)$ for $t \in \mathbb{R}_+$ and $W(uJ) = W(u) + W(J)$ for $J \in \mathcal{J}$.*

We restrict our attention to request streams that do not ask for more work per time unit than the platform can provide, and, furthermore, do not present an unbounded number of requests in a bounded time interval.

Definition 6 (Admissible, Critical and Subcritical Request Streams)**.**
A request stream σ is α-lax ($\alpha \in \mathbb{R}_+$) with respect to an execution platform R if for every $t < t'$, $W(\sigma_{[0,t']}) - W(\sigma_{[0,t]}) \leq \alpha(t' - t)R + b$ for some constant $b \in \mathbb{R}^n$. A stream is admissible if it is α-lax for some $\alpha \leq 1$, subcritical if it is α-lax for $\alpha < 1$ and critical if it is admissible but not subcritical.

Those notions are lifted to generators (set of streams). We add the keyword "uniformly" if α and b have the same value for all the streams of the set.

2.2.4 Schedules

Definition 7 (Schedule)**.** *A schedule is a function $s : T \times \mathbb{N} \to \mathbb{R}_+^\infty$ (where $\mathbb{R}_+^\infty = \mathbb{R}_+ \cup \{\infty\}$ with the usual extension of the order and operations).*

The intended meaning of $s(a, i) = t$ is that the i-th instance of task a (which is part of the i-th instance of the job type to which it belongs) starts executing at time t. If we restrict ourselves to "non-overtaking" schedules[2] such that $s(a, i) \leq s(a, i')$ whenever $i < i'$, we can view a schedule as a timed ω-word in $\mathcal{T}_\omega(T)$. Likewise we can speak of finite prefixes $s_{[0,t]}$ which are timed words in $\mathcal{T}(T)$.

Since tasks have fixed durations and cannot be preempted, a schedule determines uniquely which tasks are executed at any point in time and, hence, how many resources of each type are utilized, a notion formalized below.

[2]Note that non-overtaking applies only to tasks of the *same type*.

Definition 8 (Utilization Function, Work Supplied)**.** *The resource utilization function associated with every resource m is $U_m : \mathcal{T}_\omega(T) \times \mathbb{R}_+ \to \mathbb{N}$ defined as $U_m(s,t) = |\{(a,i) \in T \times \mathbb{N} : \mu(a) = m \wedge s(a,i) \leq t < s(a,i) + d(a)\}|$. The work supplied by a prefix of s is the accumulated utilization: $W(s_{[0,t]}) = \int_0^t U(s,\tau)d\tau$.*

Definition 9 (Valid Schedule)**.** *A schedule s is valid for a request stream σ on an execution platform R if for any task instance (a,i)*

- *if J is the job type a belongs to, then $s(a,i) \geq \sigma(J,i)$ (no proactivity: jobs are executed after they are requested);*
- *$\forall a', a' \prec a, s(a,i) \geq s(a',i) + d(a')$ (job precedences are met);*
- *$\forall t \in \mathbb{R}_+, U(s,t) \leq R$ (no overload: no more resource instances of a type are used than their total amount in the execution platform).*

The quality of a schedule can be evaluated in two principal and related (but not equivalent) ways, the first of which does not look at individual job instances but is based on the amount of work. During every prefix of the schedule there is a non-negative difference between the amount of work that has been requested and the amount of work that has been supplied. This difference can be defined in a "continuous" fashion like $\Delta_{\sigma,s}(t) = W(\sigma_{[0,t]}) - W(s_{[0,t]})$. An alternative that we will use, is based on the concept of *residue* or backlog, which is simply the set of requested tasks that have not yet started executing. It is not hard to see that a bounded residue is equivalent to a bounded difference between requested and supplied work.

Definition 10 (Residue, Bounded Residue Schedules)**.** *The residue associated with a request stream σ and a valid schedule s at time t is $\rho_{\sigma,s}(t) = \{(a,i) \in T \times \mathbb{N} : \sigma(a,i) \leq t < s(a,i)\}$. A valid schedule s is of bounded residue if there is a number c such that $|\rho_{\sigma,s}(t)| \leq c$ for every t.*

The second performance measure associated with a schedule is related to latency, the time an individual job has to wait between being requested and the completion time of its last task.

Definition 11 (Latency)**.** *Given a request stream σ and a valid schedule s, the latency of a job instance (J,i) is $L_{J,i}(\sigma,s) = \max_{a \in T_J}\{(s(a,i) + d(a))\} - \sigma(J,i)$. The latency of s with respect to σ is $L(\sigma,s) = \sup_{J \in \mathcal{J}, i \in \mathbb{N}} L_{J,i}(\sigma,s)$.*

Note that it is possible that every job instance is served in finite time but the latency of the schedule is, however, infinite, that is, the sequence $\{L_{J,i}\}_{i \in \mathbb{N}}$ may diverge. Bounded residue does not imply bounded latency: we can keep one job waiting forever, while still serving all the others without accumulating backlog. But the implication holds in the other direction.

Lemma 1. *A valid schedule with bounded latency has a bounded residue.*

Proof. Let s be a valid schedule with latency $\lambda \in \mathbb{R}_+$. Let $V(t)$ be the total amount of work of the tasks that are in the residue at time t. Since all these tasks are supposed to be completed by $t + \lambda$ we have $V(t) \leq \lambda R$ which implies a bound on the residue. □

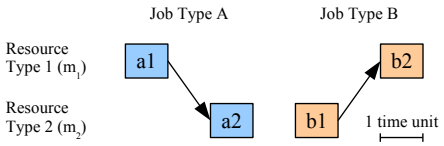


Figure 2.1: The example.

2.2.5 The Running Example

We will use the following recurrent scheduling problem to establish the negative result and to illustrate our scheduling policy. Consider a platform over $M = \{m_1, m_2\}$ with $R(m_1) = R(m_2) = 1$. The set of job types is $\mathcal{J} = \{A, B\}$ whose respective sets of tasks $\{a_1 \prec a_2\}$ and $\{b_1 \prec b_2\}$ have all a unit duration. The difference between these job types is that A uses m_1 before m_2 while B uses m_2 before m_1 (see Fig. 2.1). As a request generator we consider $G = ((A1) + (B1))^\omega$, that is, every unit of time, an instance of either one of these jobs is requested (to simplify notations we will use henceforth A and B as a shorthand for $A1$ and $B1$, respectively). Since each job type requires exactly the amount of work offered by the platform, G is admissible and, in fact, critical. A bounded-residue schedule for such critical request streams should keep the machines busy *all the time* except for some intervals (that we call *utilization gaps*) whose sum of durations is bounded.

The reversed order of resource utilization in A and B renders these two job types *incompatible* in the sense that it is not easy to "pipeline" them on our platform. Intuitively at the moment a request stream switches from A to B, we may have tasks a_2 and b_1 ready for execution but only one instance of their common resource m_2 is free. Our scheduling policy will, nevertheless, manage to pipeline them but, as we show in the next section, bounded latency schedules are impossible.

2.3 Negative Result

In this section, we show that admissibility of a stream does not imply that it is schedulable under bounded latency. We prove this fact using a counter-example stream constructed by iterating Lemma 2 below.

This lemma states that for any latency λ, we can find a certain request pattern of the running example whose occurrence implies a unit increase in the residue. Hence the construction of the counter-example stream will involve repeating this pattern infinitely many times, implying an unbounded residue. The statement of the lemma and its proof are illustrated in Fig. 2.2.

Lemma 2. *Let σ be a request stream with a prefix of the form $\sigma_{[0,t]} = uA^\lambda B^\lambda$ and let s be a valid schedule for σ with latency λ. Then there is a utilization gap (an idle resource) of duration 1 or more in the interval $[t - \lambda - 1, t]$.*

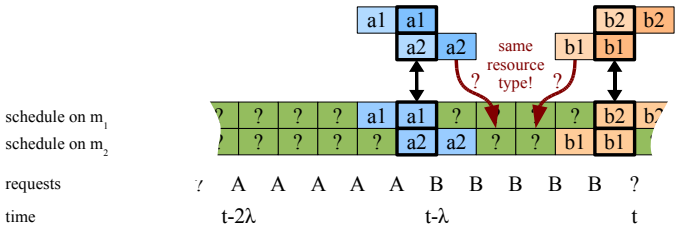


Figure 2.2: An illustration of the fact that a request segment $A^\lambda B^\lambda$ implies a utilization gap in any schedule of latency λ or less. Before $t-\lambda$: job type A has been requested for a long time, so the residue contains only tasks from A. At $t-\lambda$: from now on, requests are of type B. After $t-1$: if the latency is λ, there should be no more tasks from A in the residue. Now between $t-\lambda$ and $t-1$, only suffixes of A and prefixes of B can be scheduled, and among those, at least one proper suffix.

Proof. Since the latency of s is λ, no task instance of B belongs to the residue $\rho_{\sigma,s}(t-\lambda-1)$, so the only way to avoid a gap at time $t-\lambda-1$ is to schedule an instance of a_1 and an instance of a_2. For the same reason, $\rho_{\sigma,s}(t-1)$ contains no task instance of A, so that at time $t-1$, s schedules b_1 and b_2. Moreover any task instance of B in the residue after $t-\lambda-1$ is an instance that was requested since $t-\lambda$.

Now what happens in $[t-\lambda, t-1]$? In that interval, the residue has task instances from requests for A made before $t-\lambda$ and from requests for B made since that time. Due to bounded latency all the instances from A are due for $t-1$. We also know that, because $a_1 \prec a_2$, the residue has always more a_2 than a_1, and that their amount is the same only when all started job instances of A are finished, which is not possible at $t-1$ because an a_1 is scheduled for $t-\lambda-1$ (and thus task a_2 of the same job instance cannot start before $t-\lambda$). In that interval we also schedule task instances from B the earliest of which can have started execution at $t-\lambda$. Thus, since $b_1 \prec b_2$, we cannot schedule more b_2 than b_1.

Summing up the quantity of work scheduled by s between $t-\lambda$ and t, we find that on m_1 we schedule n_{a_1} instances of a_1 and n_{b_2} instances of b_2 and on m_2 we schedule n_{a_2} instances of a_2 and n_{b_1} instances of b_1, satisfying $n_{a_2} > n_{a_1}$ and $n_{b_1} \geq n_{b_2}$. Thus m_2 performs at least one unit of work more than m_1 in the same interval, which is only possible if m_1 admits a utilization gap of duration 1. □

Consider now a request stream that has infinitely many occurrences of the pattern $uA^\lambda B^\lambda$. A schedule with latency λ for this stream will have infinitely many gaps, and hence an unbounded residue, a fact which contradicts Lemma 1. Hence such a stream admits no schedule whose latency is λ or less.

Theorem 1. *Let*

$$L_\infty = \mathcal{J}^* ABB \mathcal{J}^* AAABBBB \mathcal{J}^* AAABBBs,$$

where $\mathcal{J}$ stands for $(A+B)$. Request streams in L_∞ admit no bounded-latency schedule, although they are admissible.

Proof. Let σ be a stream of L_∞. For every λ, σ has infinitely many prefixes of the form $uA^\lambda B^\lambda$ and cannot have a schedule of latency λ. Consequently it admits no bounded latency schedule. □

Note that this impossibility result is not related to the dynamic aspect of the scheduling problem. Even a clairvoyant scheduler who knows the whole request stream in advance cannot find a bounded latency solution.

Note also that the language L_∞ is not pathological. If fact, in any reasonable way to induce probabilities on $(A + B)^\omega$, this language will have probability of 1. Hence we can say that critical systems having two incompatible jobs will almost surely admit only unbounded-latency schedules.

2.4 Scheduling Policies

Now we want to consider the act of scheduling as a dynamic process where a scheduler has to adapt its decisions to the evolution of the environment, here the incoming request stream. We want the scheduler to construct a schedule *incrementally* as requests arrive. The mathematical object that models the procedure of mapping request stream prefixes into scheduling decisions is called a scheduling *policy* or a *strategy*.

Formally speaking, a policy can be viewed as a timed transducer, a causal function $p : \mathcal{T}_\omega(\mathcal{J}) \to \mathcal{T}_\omega(T)$ which produces for each request stream σ a valid schedule $s = p(\sigma)$. Causality here means that the value of $s_{[0,t]}$ depends only on $\sigma_{[0,t]}$. We will represent the policy as a procedure p which, at each time instant t, looks at $\sigma_{[0,t]}$ and selects a (possibly empty) set of task instances to be scheduled for execution at time t, that is, $s(a, i) = t$ if $(a, i) \in p(\sigma_{[0,t]})$. We will use $s_{[0,t]} = p(\sigma_{[0,t]})$ to denote the schedule prefix constructed by successive applications of p during the interval $[0, t]$. We assume that each policy is designed to work with admissible request streams taken from a generator $G \subseteq \mathcal{T}_\omega(\mathcal{J})$.

Definition 12 (Scheduling policy)**.** *A scheduling policy is a function $p : \mathcal{T}(\mathcal{J}) \to 2^{T\times\mathbb{N}}$ such that for every task instance (a, i) and a request stream prefix σ, $(a, i) \in p(\sigma)$ implies that $(a, i) \notin p(\sigma')$ for any $\sigma' \sqsubset \sigma$. A scheduling policy is valid for σ if for every t, the obtained schedule $s_{[0,t]} = p(\sigma_{[0,t]})$ satisfies the conditions of Definition 9, namely, no proactivity and adherence to precedence and resource constraints.*

We evaluate the overall performance of a policy based on the worst schedule it produces over the streams in the generator. Since we have just shown a negative result concerning latencies, we focus on the residue.

Definition 13 (Bounded Residue Policies)**.** *A scheduling policy has a bounded residue relative to a generator G if it produces a bounded-residue schedule for every $\sigma \in G$. It is uniformly bounded if the same constant can bound the residues of all its streams.*

In the following, we use notation $\rho_{\sigma,p}$ instead of $\rho_{\sigma,p(\sigma)}$ to denote the residue resulting from the application of a policy p to a request stream (or prefix) σ.

2.5 Positive Result

In this section we show that any recurrent scheduling problem with an admissible request generator admits a policy in the sense of Sect. 2.4 which maintains the residue bounded. We emphasize again that the policy makes decisions at run time without knowing future requests.

2.5.1 Oldest-First Policy does not Work

To appreciate the difficulty, let us consider first a naive Oldest-First policy: whenever the number of tasks that are ready to use a resource is larger than the number of free instances of the resource, the available instances are granted to the older tasks among them. We show that this policy fails to guarantee bounded residues.

Theorem 2. *The Oldest-First policy cannot guarantee a bounded residue.*

In fact, this policy will lead to an unbounded residue schedule for request streams in the language L_∞ of the previous section as illustrated in Fig. 2.3 and proved below. The reason is, again, the incompatibility between the job types, which leads to infinitely many utilization gaps where a resource is free while none of the corresponding tasks in the residue is ready to utilize it. The result is a direct corollary of the following lemma:

Lemma 3. *A bounded residue schedule which conforms to the Oldest-First policy has a bounded latency.*

Note that we already proved the converse for arbitrary schedules and policies.

Proof. First we show that any task instance (a, i) that becomes eligible for execution at time t, is scheduled for execution within a bounded amount of time after t. This holds because, following the policy, the only tasks that can be executed between t and $s(a, i)$ are those that are already in the (bounded) residue at time t. Next we show, by induction on the rank of the tasks, that this fact implies that any task is executed within a bounded amount of time after its job is issued. This holds trivially for the initial tasks which become eligible for execution immediately when the job arrives and then holds for tasks of rank $n + 1$ by virtue of the bounded latency of tasks of rank n. Thus the latency of a bounded-residue schedule produced by the Oldest-First has to be bounded. □

Since we have already shown that request streams in L_∞ do not admit bounded-latency schedules, a bounded residue strategy will lead to a contradiction and this proves Theorem 2. Like the case for Theorem 1, under reasonable probability assignments to jobs, one can show that the Oldest-First policy will almost surely lead to unbounded-residue schedules when applied to critical streams of incompatible jobs.

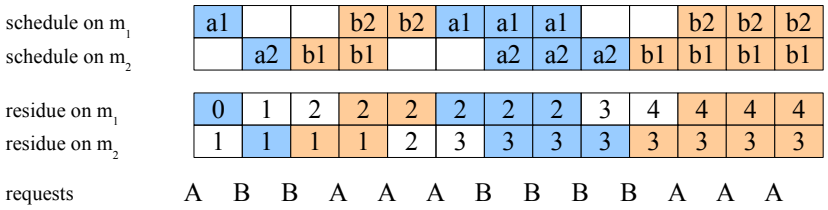


Figure 2.3: Schedule generated by the "oldest first" policy on a stream in the language L_∞, described in 2.3. Here we see that a gap of length 2 is created on one of the resource types at every change of job type in the request stream, which makes the residue grow indefinitely.

2.5.2 A Bounded Residue Policy

Theorem 3 (Bounded Residue Policy)**.**

- *Any admissible generator admits a bounded residue scheduling policy.*
- *Any uniformly admissible generator admits a uniformly bounded residue scheduling policy.*

In order to circumvent the shortcomings of the "Oldest First" policy, we describe in the sequel a policy that eventually reaches the following situation: whenever a resource becomes free and the residue contains tasks that need it, at least one of those tasks will be ready for execution.

The policy is described in detail in Algorithm 1 and We explain the underlying intuition below. The policy separates the act of choosing which tasks to execute in the future from the act of actually starting them. The first decision is made upon job arrival while the second is made whenever a resource is free and a corresponding task has been selected. To this end we partition the residue into two parts. The first part P (the "pool") consists of requested task instances that have not yet been selected for execution. Among those, only task instances whose $\prec$-predecessors have already terminated are eligible for being selected and moved to the other part, which consists of n FIFO queues $\{Q_m\}_{m\in M}$, one for each resource type. The passage between the two is controlled by two types of events:

- Task termination: when a task (a, i) terminates, eligibility status of its successors in P is updated;
- Job arrival: when a job instance (J, i) arrives we pick the oldest[3] eligible instance $(a, j_a) \in P$ (if such exists) for every task type $a \in T_J$ such that $\mu(a) = m$, and move it to Q_m. Note that only initial tasks of (J, i) are eligible for being selected when (J, i) arrives, while for other task types only earlier instances can be chosen.

[3]Or one of the oldest if there are several of the same age.

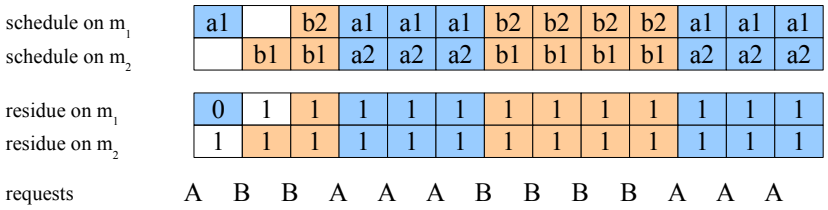


Figure 2.4: The schedule generated by the bounded-residue policy for σ_∞. We can see that after the arrival of the second B, every resource is always occupied, and that the residue does not grow after that.

Whenever a resource of type m is free and Q_m is not empty, the first element is removed from Q_m and starts executing. This is sufficient to ensure a bounded residue. However, to improve the performance of the algorithm when the streams are subcritical, we also choose to start the oldest eligible task which requires m if an instance of m is released when Q_m is empty.

The intuition why this policy works is easier to understand when we look at critical request streams. For such streams, any job type which is requested often enough will eventually have instances of each of its tasks in Q and hence, whenever a resource is freed, there will always be some task ready to use it. This guarantees smooth pipelining and bounded residue for all admissible request streams. In Fig. 2.4 we can see how our policy schedules the request stream $\sigma_\infty = A \cdot B \cdot B \cdot A \cdot A \cdot A \cdots$.

Scheduling policies of the FIFO type have also been studied in the context of adversarial queuing and it has been shown under various hypothesis [26] that those were not stable, sometimes even for arbitrarily small loads. What makes our policy work is the fact that the act of queuing is triggered by a global event (arrival of a new job request) on which the actual choice of tasks to be queued depends. So the decision is somehow "conscious" of the global state of the system, as opposed to what happens in a classical FIFO network.

Now we prove that Algorithm 1 yields a valid schedule with (uniformly) bounded residue for any (uniformly) admissible generator, which establishes Thm. 3. To this end we need to introduce 2 lemmas.

Lemma 4. *If $\sigma \in G$ is b-admissible, then $\forall m \in M, \forall t \in \mathbb{R}_+, |Q_m(t)| \leq b_m + D \cdot R_m$, where D is the maximal length of a task type, and where $|Q_m|(t)$ is the size of Q_m at time t.*

Proof. In fact, the size of a queue at a time t is the difference between the enqueued work and the dequeued work, hence we know how to relate those quantities to the dimensions of the platform.

It is easy to see that when the policy is applied, an instance of a resource m can never stay free for longer than 0 units of time unless Q_m is empty at the time it is freed. Therefore we just have to prove that the Q_m cannot grow arbitrarily during intervals

Algorithm 1 The Bounded-Residue Policy

```
declarations
    req: jobType inputEvent                                   // events from σ
    free: resourceType inputEvent                  // triggered when a resource is freed
    start: taskInstance outputEvent                           // scheduling decisions
    P: taskInstance set                                       // pool, unselected tasks
    Q: resourceType → (taskInstance fifo)                     // queues, selected tasks

procedure INIT
    P = ∅;
    for all m ∈ M do Q_m = ∅

procedure STARTWORK(m: resourceType)
    if Q_m is not empty then α = pop(Q_m); emit start(α)
    else                                  // for bounded latency against subcritical streams
        if P has eligible task instances requiring m then
            α = the oldest eligible task instance using m in P;
            P = P − {α}; emit start(α)

on free(m) do startWork(m)

on req(J) do
    for all a ∈ T_J do
        P = P ∪ {newInstance(a)};
        if P has eligible task instances of type a then
            α = the oldest eligible task instance of a in P;
            P = P − {α}; push(α, Q_m)                         // select for execution
    for all m ∈ M do
        for all free instances of m do startWork(m)
```

where $U_m(s,t) = R_m$ (intervals of length 0 do not matter as the utilization function U is constant on semi-open intervals).

Let $[t_0, t_1)$ be an interval during which the utilization is full. The amount of work dequeued during $[t_0, t_1)$ is at least the capacity available in the interval $[t_0, t_1)$ minus the maximum amount of work that may already have been executing at the beginning of the interval. I.e., if D denotes the length of the longest task type, then we have: $W_m(s^{-1}([t_0, t_1))) \geq (t_1 - t_0)R_m - DR_m = (t_1 - t_0 - D)R_m$.

The quantity of work that is enqueued during that interval is at most the quantity of work of the request word of the interval, so it is at most $W(\sigma_{[0,t_1]}) - W(\sigma_{[0,t_0]})$. Since σ is b-admissible, the following holds:

$$\begin{aligned} W_m(\sigma_{[0,t_1]}) - W_m(\sigma_{[0,t_0]}) &\leq (t_1 - t_0) \cdot R_m + b_m \\ &= (t_1 - t_0 - D) \cdot R_m + b_m + D \cdot R_m \\ &\leq W_m(s^{-1}([t_0, t_1))) + b_m + D \cdot R_m. \end{aligned}$$

Therefore, in any interval where the queue is not empty, the quantity of work in the queue can only increase by at most $b_m + D \cdot R_m$, which is independent of the length of the interval. Thus the quantity of work in the queue for m is never more than $b_m + D \cdot R_m$.

Since there exists $d \in \mathbb{R}_+ - \{0\}$ such that all tasks last more than d, there are at most $(b_m + D \cdot R_m)/d$ tasks in Q_m. □

Lemma 5. *For any $\sigma \in G$, there is a time T after which at every request for a job J, the union of the set of tasks we put in Q at that time and of the set of task instances from task types of J that have been directly started as a "fill up" since previous request for J, contains exactly one instance of each task type of J.*

Proof. Let us say a task instance is *selected* for a request if it falls under the above criteria, that is if it is an element of the union described in the statement of the lemma.

Base case: for the initial tasks this is trivially true, since they are queued as soon as the job is requested.

Inductive case: now suppose there exists a time t_{l-1} after which when a job of type J is requested, a task instance of each type of rank $l-1$ is selected for that request. Since the policy chooses the oldest instance first, the difference of indices between the job instance from which the task comes, and the job request at which it is selected is bounded. And after it is queued (if it is), it will be served in bounded time, during which the number of requests for J is also bounded. Thus any requested task of rank $l-1$ is served after a bounded number of requests for J. Let N_{l-1} be that number.

Let a be a task type of rank l from job type J. Then an instance of a has to become eligible before N_{l-1} requests for J after its own. This we can write as $n_r(t) - N_{l-1} \leq n_e(t)$, where $n_r(t)$ is the number of requested J at to time t and $n_e(t)$ is the number of tasks instances form a made eligible since the beginning.

Let $n_s(t)$ be the number of selected instances of a until time $t \in \mathbb{R}_+$. Then there are two possibilities:

- Either the number of eligible but not selected instances of a stays smaller than N_{l-1} ($n_e(t) - n_s(t) \leq N_{l-1}$), so that any of them has to be selected before N_{l-1} requests for J. Thus $n_s(t) + N_{l-1} \geq n_e(t) \geq n_r(t) - N_{l-1}$, which gives $n_r(t) - n_s(t) \leq 2N_{l-1}$. Hence $n_r(t) - n_s(t)$ never decreases (we select no more than requested), therefore it is eventually constant, and thus eventually we queue an instance of a every time a J is requested.

- Or $n_e(t) - n_s(t)$ can go above $N_{l-1} + 1$. Then, suppose it is true at a time t_0.

  Then we prove that $n_e(t) - n_s(t)$ can never go to zero after t_0: indeed, $n_s(t)$ increases only when $n_r(t)$ does, so that means that $\forall t \geq t_0, n_s(t) - n_s(t_0) \leq n_r(t) - n_r(t_0)$, so $n_s(t) \leq n_r(t) - n_r(t_0) + n_s(t_0) \leq n_r(t) - n_r(t_0) + n_e(t_0) - N_{l-1} - 1 \leq n_e(t) - n_r(t_0) + n_e(t_0) - 1$, and thus $n_e(t) - n_s(t) \geq n_r(t_0) - n_e(t_0) + 1 > 1$.

  So after t_0 there are always eligible instances of a that are not yet selected, so after t_0, when a job is requested, an instance of each of its tasks of rank l is selected.

In both cases, the fact that eventually at every request we queue an instance of every task type of rank $l-1$ implies that we eventually also select one instance for each type of rank l. That means, as there are only finitely many ranks, that eventually it is true for any rank. □

Proof. of Thm. 3

Validity of the produced schedules Let σ be a request stream and let s be the schedule for σ produced by p. Then, for any task instance $\alpha = (a, i)$ s.t. $s(\alpha) = t$:

- α had been in the pool at some previous $t' < t$. We therefore know that s is not proactive, because the pool contains only requested job instances.

- Either α was in a queue at t. Hence, as a task gets moved from the pool to a queue only if all of its predecessors have finished, s respects dependencies. Or α has been started when the queue was empty, and in that case the algorithm states that α was ready (its predecessors were all finished).

- Execution of α is started only in procedure *startWork*(m). That procedure starts executing only a single task instance and is called at most once for each free resource. Hence, s does not exceed resource capacities.

These 3 conditions prove that s is a valid schedule.

Bounded Residue Using Lemma 4, we establish that the queues are bounded. After that we show boundedness of the pool by proving the following invariant (Lemma 5): we eventually enqueue, at every request for a job type, one instance of every task type that belongs to that job type.

Conclusion (the residue is thus bounded): Lemma 5 states that eventually we select, at each request for a job type J, one task instance per task type from J, so that the total amount of task instances of each type in the pool cannot increase anymore. Hence the residue is the union of the pool and of the queues, which are also bounded (Lemma 4), thus the residue is bounded. That proves the first part of Thm. 3.

If we look at the proof, we notice that the bound on the queues and the maximal time at which the property of Lemma 5 comes true depend on the request stream σ only through the constant b that characterizes its admissibility. Thus if G is uniformly admissible, then the residue is uniformly bounded, which proves the second part of the theorem.

□

2.5.3 Bounded Latency for Subcritical Streams

We just showed that a policy could ensure bounded residues in the case of critical streams for which one needs full utilization. But criticality is just a limit case and for that reason it is interesting to know whether such a policy can adapt and behave better when the request stream is subcritical. Fortunately the answer is positive: the previously exhibited policy, by starting tasks which are not queued when a resource would be otherwise idle, ensures bounded latencies for request streams that admit some laxity.

Theorem 4. *The policy described by Algorithm 1 has a bounded latency when applied to any α-lax stream with $\alpha < 1$.*

Lemma 6. *There exists a time bound $T_{\alpha,m}$ such that any interval $[t, t+T_{\alpha,m}]$ admits a time instant where Q_m is empty, an instance of m is free and no new request arrives.*

Sketch of proof. Consider an interval of the form $[t, t+d]$ in which no machine of type m is idle. The quantity of work dequeued from Q_m is $R_m d$ and, due to laxity, the amount of work enqueued into Q_m is at most $(1-\alpha)R_m d$. Hence the total contribution to the amount of work in Q_m is $(\alpha-1)R_m d$ and for some sufficiently large d it will empty Q_m. □

Proof. of Theorem 4]
We know that, when a task in the pool becomes the oldest task of the residue which is not queued, it becomes eligible in a bounded amount of time (all its predecessors must be in the queue). Thus we know that at most $T_{\alpha,m}$ units of time after that, this task is started (either queued or started to fill a gap). Since furthermore the residue (and hence the pool) is bounded (Thm. 3), there is a bound on the time it takes a task to become the oldest in the pool and hence to be executed. Thus we conclude that the latency of the policy is bounded. □

2.6 Discussion

We have proved some fundamental results on a model that captures, we believe, many real-world phenomena. Let us mention some related attempts to treat similar problems. The idea that verification-inspired techniques can be used to model and then solve scheduling problems that are not easy to express in traditional real-time scheduling models has been studied within the timed controller synthesis framework and applied to scheduling problems [52, 43, 9, 1]. What is common to all these approaches (including [30] which analyzes *given* policies that admit task preemption) is that the scheduler is computed using a verification/synthesis algorithm for timed automata, which despite several improvements [21] are intrinsically not scalable. The policy presented in this chapter does not suffer from this problem, it only needs the request generator to be admissible. Explicit synthesis may still be needed in more complex settings.

In the future it would be interesting to investigate various extensions of the model and variations on the rules of the game, in particular, moving from worst-case reasoning to average case by using probabilistic request generators and evaluating policies according to expected backlog or latency. Finally, it would be interesting to look closer at the question of "pipelinability", that is, the mutual compatibility of a set of job types. Results in this direction may lead to new design principles for request servers.

Chapter 3

On ω-Languages Defined by a Mean-Payoff Conditions

3.1 Introduction

In the previous chapter of this thesis, we proved some results about what was possible in terms of latency and memory usage for a scheduling policy. Latency and memory are just two *quantitative* measures we can associate with a behavior (run) of a system and one can think of many other measures (for example, power consumption) that can be associated with such behaviors, reflecting their respective cost and utility. The average values that a run obtains for these cost criteria can be used as an alternative way to classify it as accepted or rejected. The translation of this scheduling problem to automata is natural, even more now that it is established that it can be encoded into a finite number of states. It is therefore tempting to use standard automata verification techniques.

In algorithmic verification of reactive systems, the system is usually modeled as a finite-state transition system (possibly with fairness constraints), and requirements are captured as languages of infinite words over system observations [47, 44]. The most commonly used framework for requirements is the class of ω-regular languages. This class is expressive enough to capture many natural requirements, and has well-understood and appealing theoretical properties: it is closed under Boolean operations, it is definable by finite automata (such as deterministic parity automata or nondeterministic Büchi automata), it contains Linear Temporal Logic (LTL), and decision problems such as emptiness, language inclusion are decidable [51, 50, 45].

However the classical verification framework only captures *qualitative* aspects of system behavior, and does not work well with *quantitative* aspects like consumption of resources such as CPU and energy and, obviously, average performance criteria in scheduling. In order to describe these, a variety of extensions of system models, logics, and automata have been proposed and studied in recent years [39, 25, 23, 5]. The best known approach, and the most relevant to our work, is the following: a *payoff* (or a cost) is associated with each state (or transition) of the model, the *mean-payoff* of a

finite run is simply the average of the payoffs along the run, and the mean-payoff of an infinite run is the limit, as n goes to infinity, of the mean-payoff of the prefix of length n. The notion of mean-payoff objectives was first studied in classical game theory, and more recently in verification literature [53, 25, 32]. Most of this work is focused on computing the optimal mean-payoff value, typically in the setting of two-player games, and the fascinating connections between the mean-payoff and parity games.

In this chapter, we propose and study ways of defining languages of infinite words based on mean-payoff criteria. As a motivating example, suppose 1 denotes the condition "message is delivered" and 0 denotes the condition "message is lost." A behavior of the network is an infinite sequence over $\{0, 1\}$. Requirements such as "no message is ever lost" (always 1), "only finitely many messages are lost" (eventually-always 1), and "infinitely many messages are delivered" (infinitely-often 1), are all ω-regular languages. However, the natural requirement that "the number of lost messages is negligible" is not ω-regular. Such a requirement can be formally captured if we can refer to *averages*. For this purpose, we can associate a payoff with each symbol, payoff 0 with message lost and payoff 1 with message delivered, and require that the mean-payoff of every infinite behavior is 1. As this example indicates, using mean-payoffs to define acceptance conditions can express meaningful, non-regular, and yet analyzable, requirements.

The central technical question for this chapter is to define a precise query language for mapping mean-payoffs of infinite runs into Boolean answers so that the resulting class of ω-languages has desirable properties concerning closure, expressiveness, and analyzability. The obvious candidate for turning mean-payoffs into acceptance criteria is threshold queries of the form "is mean-payoff above (or below) a given threshold θ". Indeed, this is implicitly the choice in the existing literature on decision problems related to mean-payoff models [53, 25, 32]. A closer investigation indicates that this is not a satisfactory choice for queries, in particular for the scheduling problem that was one of the motivations of this work.

In particular, this approach only models phenomena where a single quantitative criterium is taken into account, as we show that the intersection of several ω-languages belonging to that class cannot be expressed with a weighted automaton and a single mean-payoff condition. Closure under intersection actually requires that we should be able to model multiple payoff functions. For this purpose, we define *d-payoff automata*, where d is the dimension of the payoffs, and each edge is annotated with a d-dimensional vector of payoffs. We prove that expressiveness strictly increases with the dimension. From the applications point of view, multi-payoffs allow to model requirements involving multiple quantities. Because we allow unbounded dimension, one can also add coordinates that model weighted sums of the quantities, and put bounds on these coordinates too.

Second, the limit of the mean-payoffs of prefixes of an infinite run may not exist. This leads us to consider the set of *accumulation points* corresponding to a run. For one-dimensional payoffs, the set of these points is an interval. For multi-dimensional payoffs, we are not aware of existing work on understanding the structure of these points. We establish a precise characterization of the structure of accumulation points:

a set can be a set of accumulation points of a run of a payoff automaton if and only if it is closed, bounded, and connected.

Third, if we use mp to refer to the mean-payoff of a run, and consider four types of queries of the form $mp < \theta$, $mp \leq \theta$, $mp > \theta$, and $mp \geq \theta$, where θ is a constant, we prove that the resulting four classes of ω-languages have incomparable expressiveness. Consequently acceptance conditions need to support combinations of all such queries.

After establishing a number of properties of the accumulation points of multi-dimensional payoff automata, we propose the class of *multi-threshold mean-payoff languages*. For this class, the acceptance condition is a Boolean combination of constraints of the form "is there an accumulation point whose i^{th} projection is less than a given threshold θ". We show that the expressive power of this class is incomparable to that of the class of ω-regular languages, that this class is closed under Boolean operations and has decidable emptiness problem. We also study its Borel complexity.

These results were first published in [2].

3.2 Definitions

3.2.1 Multi-Payoff Automata

Multi-payoff automata are defined as automata with labels, called payoffs, attached to transitions. In this chapter, payoffs are vectors in a finite dimensional Euclidean space.

Definition 14 (d-Payoff automaton)**.** *A d-payoff automaton, with $d \in \mathbb{N}$, is a tuple $\langle A, Q, q_0, \delta, w\rangle$ where A and Q are finite sets, representing the alphabet and states of the automaton, respectively; $q_0 \in Q$ is an initial state; $\delta \in Q \times A \to Q$ is a total transition function (also considered as a set of transitions $(q, a, \delta(q, a))$) and $w\colon \delta \to \mathbb{R}^d$ is a function that maps each transition to a d-dimensional vector, called payoff.*

Note that we consider only deterministic complete automata.

Definition 15. *The following notions are defined for payoff automata:*

- *A finite* run *of an automaton is a sequence of transitions of the following type: $(q_1, a_1, q_2)(q_2, a_2, q_3)\dots(q_i, a_i, q_{i+1})$. An infinite run is an infinite sequence of transitions such that any prefix is a finite run.*
- *We denote by $\lambda(r)$ the word of the symbols labelling the successive transitions of the run r, i.e. $\lambda((q_1, a_1, q_2)\cdots(q_n, a_n, q_{n+1})) = a_1 \cdots a_n$.*
- *A run is* initial *if $q_1 = q_0$.*
- *By $\mathrm{run}_{\mathcal{A}}(u)$ we denote the initial run r in $\mathcal{A}$ such that $u = \lambda(r)$*
- *A* cycle *is a run $(q_1, a_1, q_2)(q_2, a_2, q_3)\dots(q_i, a_i, q_{i+1})$ such that $q_1 = q_{i+1}$. A cycle is* simple *if no proper subsequence is a cycle.*

- *For a word or run u, $u\restriction n$ denotes the prefix of length n of u, and $u[n]$ the n^{th} element of u.*
- *The payoff of a finite run r is* $\mathrm{payoff}(r) = \sum_{i=1}^{|r|} w(r[i])$.
- *The mean-payoff of a run r is* $\mathrm{mp}(r) = \mathrm{payoff}(r)/|r|$.
- *A subset of the states of an automaton is* strongly connected *if, for any two elements of that subset, there is a path from one to the other.*
- *A* strongly connected component *(SCC) is a strongly connected subset that is not contained in any other strongly connected subset.*
- *A SCC is* terminal *if it is reachable and there is no path from the SCC to any other SCC.*

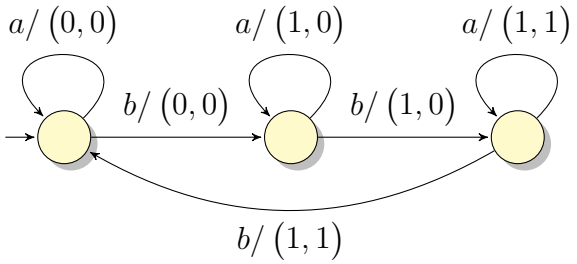
3.2.2 Acceptance

In the literature, the mean-payoff value of a run is generally associated with the "limit" of the averages of the prefixes of the run. As that limit does not always exist, standard definitions only consider the lim inf of that sequence (or sometimes lim sup) and, more specifically, threshold conditions comparing those quantities with fixed constants [53, 25, 22, 24]. As this choice is arbitrary, and more can be said about the properties of that sequence than the properties of just its lim inf or even both its lim inf and lim sup, in particular when $d > 1$, we choose to consider the entire set of accumulation points of that sequence.

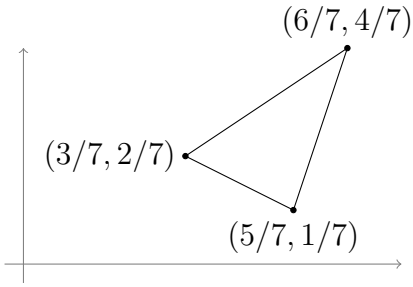
A point x is an accumulation point of a sequence $x_0, x_1, \ldots$ if every open set containing x contains infinitely many elements of the sequence.

Definition 16. *We denote by* $\mathrm{Acc}(x_n)_{n=1}^{\infty}$ *the set of accumulation points of the sequence* $(x_n)_{n=1}^{\infty}$. *If r is a run of a d-payoff automaton $\mathcal{A}$,* $\mathrm{Acc}_{\mathcal{A}}(r) = \mathrm{Acc}(\mathrm{mp}(r\restriction n))_{n=1}^{\infty}$, *and for a word w,* $\mathrm{Acc}_{\mathcal{A}}(w) = \mathrm{Acc}_{\mathcal{A}}(\mathrm{run}(w))$.

Example 1. *Consider the 2-payoff automaton*



where edges are annotated with expression of the form σ/v meaning that the symbol σ triggers a transition whose payoff is v. Let $w = \prod_{i=0}^{\infty} a^{2^i-1}b$ be an infinite word where b's are separated by sequences of a's with exponentially increasing lengths. The set $\mathrm{Acc}_{\mathcal{A}}(w)$ *is the triangle*



as we show next. By direct calculation we get that $\lim_{n\to\infty} \mathrm{mp}(w{\restriction}\sum_{i=0}^{3n} 2^i) = (6/7, 4/7)$, $\lim_{n\to\infty} \mathrm{mp}(w{\restriction}\sum_{i=0}^{3n+1} 2^i) = (3/7, 2/7)$, *and also* $\lim_{n\to\infty} \mathrm{mp}(w{\restriction}\sum_{i=0}^{3n+2} 2^i) = (5/7, 1/7)$. *Furthermore, for every* $n \in \mathbb{N}$, $j \in \{0, 1, 2\}$ *and* $k \in \{0, \ldots, 2^{3n+j+1}\}$, *the vector* $\mathrm{mp}(w{\restriction}k + \sum_{i=0}^{3n+j} 2^i)$ *is in the convex hull of* $\mathrm{mp}(w{\restriction}\sum_{i=0}^{3n+j} 2^i)$ *and* $\mathrm{mp}(w{\restriction}\sum_{i=0}^{3n+j+1} 2^i)$ *and the maximal distance between points visited on this line goes to zero as* n *goes to infinity. Together, we get that the points to which the mean-payoff gets arbitrarily close are exactly the points on the boundary of the above triangle. Similarly, if we choose the word* $w' = \prod_{i=0}^{\infty} a^{3^i-1}b$, *we get that* $\mathrm{Acc}_{\mathcal{A}}(w')$ *is the boundary of the triangle* $(4/13, 3/13), (10/13, 1/13), (12/13, 9/13)$.

We say that a word or run is *convergent*, whenever its set of accumulation points is a singleton, i.e. when its sequence of mean payoffs converges. For instance, periodic runs are convergent because the mean-payoffs of the prefixes $r{\restriction}n$ of an infinite run $r = r_1 r_2^\omega$ converge to the mean-payoff of the finite run r_2, when n goes to infinity.

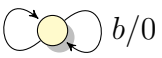
We now define acceptance conditions as as subsets of $2^{\mathbb{R}^d}$.

Definition 17. *An infinite run* r *is accepted by a* d*-payoff automaton* $\mathcal{A}$ *with condition* F, *where* F *is a subset of* $2^{\mathbb{R}^d}$, *if and only if* $\mathrm{Acc}_{\mathcal{A}}(r) \in F$. *Equivalently, we consider* F *as a predicate on* $2^{\mathbb{R}^d}$ *and for* $X \subseteq \mathbb{R}^d$, *we use the notation* $F(X)$ *for* $X \in F$.

An infinite word u *is accepted if and only if* $\mathrm{run}(u)$ *is accepted. We denote by* $L(\mathcal{A}, F)$ *the language of words accepted by* $\mathcal{A}$ *with condition* F. *In the following, we call* mean-payoff language, *any language accepted by a* d*-payoff automaton with such a condition.*

If d *is one and* $F(S)$ *is of the form* $\mathrm{extr}\, S \bowtie C$ *where* $\mathrm{extr} \in \{\inf, \sup\}$, $\bowtie \in \{<, \leq, >, \geq\}$, *and* C *is a real constant; we say that* F *is a threshold condition.*

Example 2. *For the* 1*-payoff automaton*



let the acceptance condition $F(S)$ *be true iff* $S = \{0\}$. *This defines the language of words having negligibly many* a*'s.*

3.3 Expressiveness

3.3.1 Comparison with ω-regular languages

Before proving specific results on the class of mean-payoff languages, we show that it is incomparable with the class of ω-regular languages. In this context, we call specification types incomparable if each type of specification can express properties that are not expressible in the other type. Incomparability of mean-payoff and ω-regular specifications is, of course, a motivation for studying mean-payoff languages.

We will need the following ad-hoc pumping lemma for ω-regular languages.

Lemma 7 (Pumping lemma)**.** *Let L be an ω-regular language. There exists $p \in \mathbb{N}$ such that, for each $w = u_1 w_1 u_2 w_2 \dots u_i w_i \dots \in L$ such that $|w_i| \geq p$ for all i, there are sequences of finite words $(x_i)_{i\in\mathbb{N}}, (y_i)_{i\in\mathbb{N}}, (z_i)_{i\in\mathbb{N}}$ such that, for all i, $w_i = x_i y_i z_i$, $|x_i y_i| \leq p$ and $|y_i| > 0$ and for every sequence of pumping factors $(j_i)_{i\in\mathbb{N}} \in \mathbb{N}^{\mathbb{N}}$, the pumped word $u_1 x_1 y_1^{j_1} z_1 u_2 x_2 y_2^{j_2} z_2 \dots u_i x_i y_i^{j_i} z_i \dots$ is in L.*

Proof. Similar to the proof of the pumping lemma for finite words. □

Proposition 1. *There exists a mean-payoff language, defined by a 1-payoff automaton and a threshold acceptance condition, that is not ω-regular.*

Proof. Consider the 1-payoff automaton

We show that $L = \{w | \inf \mathrm{mp}_{\mathcal{A}}(w) \leq 0\}$ is not regular. For any p, the word $w = (a^p b^{2p})^\omega$ is in that language. Assuming, towards contradiction, that the language is regular and using the pumping Lemma 7 on w, we can select as factors w_i the sequences of a an choose $j_i = 2$ to obtain a word w' that should be in L. But since $\mathrm{mp}_A(w')$ is a singleton bigger than zero, w' does not satisfy the acceptance condition and therefore is not in L, a contradiction. □

Proposition 2. *There exists an ω-regular language that is not a mean-payoff language.*

Proof. Let $L = (a^* b)^\omega$. We will show that, in any payoff automaton, we can find two ω-words u_1 and u_2, u_1 having infinitely many b and u_2 having eventually only a, and such that $Acc(u_1) = Acc(u_2)$. Then obviously no general mean-payoff acceptance condition can distinguish these two words although $u_1 \in L$ and $u_2 \notin L$.

Let us construct the counter-example. Suppose $\mathcal{A}$ is a payoff automaton recognizing L with some predicate F. Let c_a be a cycle such that $\lambda(c_a)$ contains only a's and c_b a cycle such that $\lambda(c_b)$ contains at least one b, both starting in some state q in a terminal strongly connected component of $\mathcal{A}$, and let p be an initial run leading to q.

The mean-payoffs of the run $r = p\prod_{i=1}^{\infty} c_a^i c_b$, which should be accepted, converge to $\mathrm{mp}_{\mathcal{A}}(c_a)$, which is also the mean-payoff of pc_a^ω, which should be rejected but has to be accepted by $\mathcal{A}$, since it has the same mean-payoff as r. □

3.3.2 Topology of Mean-Payoff Accumulation Points

In this section we discuss the structure of the set of accumulation points. In particular we characterize the sets that are the accumulation points of some run of a payoff automaton.

If S is a strongly connected component of an automaton, and C is the set of simple cycles in S, then we denote by $\mathrm{ConvHull}(S)$ the convex hull of $\{\mathrm{mp}(c)|c \in C\}$.

Theorem 5. *Let r be an infinite run of a d-payoff automaton, then $\mathrm{Acc}(r)$ is a closed, connected and bounded subset of $\mathbb{R}^d$.*

Proof.

Closed: True for any set of accumulation points: let (a_n) be a sequence in a topological space, and $(x_n) \in \mathrm{Acc}(a_n)_{n=1}^{\infty}$ be a sequence of accumulation points converging to a point x. For any x_i, we can choose a sub-sequence (a_{i_n}) converging to x_i. Now we can construct a sub-sequence of elements that converges to x: for every i, take the first element $a_{i_{f(i)}}$ of a_{i_n} which is at a distance smaller than 2^{-i} from x_i such that $f(i) > f(i-1)$. Then the sequence $(a_{i_{f(i)}})_{i\in\mathbb{N}}$ converges to x.

Bounded: As we are speaking of a sequence of averages of the (finite) set of payoffs, it is clear that the sequence of mean-payoffs remains in the convex hull of that set, which is bounded.

Connected: Proof by contradiction. Suppose there exists two disjoint open sets O_1 and O_2 such that $Acc(r) \subseteq O_1 \cup O_2$. Let d be the distance between O_1 and O_2. As those sets are open and disjoint, $d > 0$. But the vector between two successive averages is $\mathrm{payoff}(r\restriction n)/n - \mathrm{payoff}(r\restriction n-1)/n-1 = (1/n)(\mathrm{payoff}(r\restriction n-1)) + w(r[n]) - n/(n-1)\,\mathrm{payoff}(r\restriction n-1)) = (1/n)(w(r[n]) - \mathrm{mp}(r\restriction n))$, whose norm is smaller than Δ/n, where $\Delta = \max\{\|w(t) - w(t')\||t, t' \in \delta\}$. If a run has accumulations points in both O_1 and O_2, then there exist $n > \Delta/d$ such that the n^{th} step is in O_1 and the $(n+1)^{th}$ in O_2. The distance between those two points has to be both greater than d and smaller than Δ/n, which is not possible. □

As a remark, we can say more than boundedness: indeed a run eventually comes into a SCC it never leaves. The contribution of the payoffs of the SCC becomes then dominant as n goes to the infinity. Even better, actually, the contribution of the simple cycles of that SCC is dominant. Thus the set of accumulation points is included in the convex hull of the simple cycles of the SCC.

The following theorem is a converse to Theorem 5.

Theorem 6. *For every non-empty, closed, bounded and connected set $D \subset \mathbb{R}^d$, there is a d-payoff automaton and a run r of that automaton such that $\mathrm{Acc}(r) = D$.*

Proof. Take any automaton with a reachable SCC such that D is contained in the convex hull of the cycles of the SCC.

For every integer $n > 0$, let $\{O_{i,n} : i = 1, \ldots, l_n\}$ be a finite coverage of D by open sets of diameter smaller than $1/n$. Such a coverage exists, for example, by covering D by spheres of diameter $1/n$.

Suppose p is a finite initial run going into the SCC. For every n and every i, we can prolong p with a suffix c such that $\mathrm{mp}(pc) \in O_{n,i}$ and pc is in the SCC (form the end of p onwards). For that, we need c to be long enough and have the right proportions of simple cycles. Furthermore, as $\mathrm{mp}(pc{\restriction}l+1) - \mathrm{mp}(pc{\restriction}l)$ becomes smaller as l goes to infinity, we can make the distance of $\mathrm{mp}(pc{\restriction}l)$ from D converge to zero as l goes to infinity.

As the set $(O_{n,i})_{n,i\in\mathbb{N}\times\mathbb{N}}$ is countable, we can construct recursively the successive suffixes $c_{1,1}, c_{1,2}, \ldots, c_{2,1}, c_{2,2}, \ldots$ such that $\mathrm{mp}(pc_{1,1}c_{1,2}\ldots_{2,1}c_{2,2}\ldots c_{n,i})$ is in $O_{n,i}$, and such that for every l, $\mathrm{mp}(p\prod_{ji\in\mathbb{N}\times\mathbb{N}} c_{ji}{\restriction}l)$ is at a distance smaller than K/l from D.

Let $x \in D$. Then for every n, $x \in O_{n,i}$ for some i, thus for every n, the sequence of mean-payoffs comes within a radius $1/n$ from x, which means x is an accumulation point. Conversely, if $y \notin D$, as D is closed, it is at a distance $\delta > 0$ from D, moreover there exist a l such that $\mathrm{mp}(pc{\restriction}l)$ never exits a radius $\epsilon < \delta$ around D and therefore the sequence of mean-payoff will never come in a radius $\delta - \epsilon$ from y. So y is not an accumulation point. We conclude that $\mathrm{Acc}_{\mathcal{A}}(r)$ is exactly D. □

Actually, as for Theorem 5, a careful examination of the proof reveals that a stronger statement is true. Specifically, it is easy to verify that any closed, bounded and connected set included in any of the SCC of an automaton is the set of accumulation points of some run of that automaton.

3.3.3 Comparison of Threshold Mean-Payoff Languages

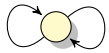
We study mean-payoff languages where the minimum and maximum of the set of accumulation points are compared with a given threshold. We assume, without loss of generality, that the threshold is zero because changing the threshold is equivalent to an affine transformation of the payoffs. We show that the different threshold languages are incomparable in expressive power.

Definition 18. *We denote by $\mathcal{L}_{\bowtie}$ the class of mean-payoff languages accepted by a 1-payoff automaton with the condition $\min \mathrm{Acc}(w) \bowtie 0$, where $\bowtie$ is $<, >, \leq$ or $\geq$.*

Note that these languages are the winning conditions used to define mean-payoff games, e.g. in [53], because $\min \mathrm{Acc}(w) = \liminf_{n\to\infty} \mathrm{mp}_{\mathcal{A}}(w{\restriction}n)$. We do not need to discuss the class of languages defined as complements of these conditions because $\mathcal{L}_{>}$ is $\mathrm{co}\,\mathcal{L}_{\leq}$ and $\mathcal{L}_{\geq}$ is $\mathrm{co}\,\mathcal{L}_{<}$, where $\mathrm{co}\,\mathcal{L}_{\bowtie}$ is the set of languages defined as sets of words that do not satisfy $\min \mathrm{Acc}(w) \bowtie 0$ for some automaton.

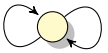
Theorem 7. *The classes $\mathcal{L}_{<}$, $\mathcal{L}_{\leq}$,$\mathcal{L}_{\geq}$ and $\mathcal{L}_{>}$ are incomparable.*

Proof. We begin by showing that $\mathcal{L}_<$ and $\mathcal{L}_\leq$ are incomparable. Consider the automaton $\mathcal{A}_1$



and the language $L_1 \stackrel{def}{=} \{w | \min \mathrm{Acc}(w) < 0\}$, which is the language of ω-words on $\{a, b\}$ having infinitely many prefixes where the proportion of as is above some constant positive value. Suppose, towards contradiction, that there exists an automaton $\mathcal{A}'_1$ accepting L_1 with a $\mathcal{L}_\leq$ condition. Consider c_a and c_b two cycles in $\mathcal{A}'_1$, labelled respectively with a word in $a(a+b)^i$ and with b^j for some integers i and j, and start from a same reachable state q (two such cycles exist at least in any terminal strongly connected component). Let p be a finite initial run ending at q. As pc_a^ω is a run of $\mathcal{A}_1$ which should be accepted, it is necessary that $\mathrm{payoff}_{\mathcal{A}'_1}(c_a) \leq 0$, and as pc_b^ω should not be accepted, it is necessary that $\mathrm{payoff}_{\mathcal{A}'_1}(c_b) > 0$. For all k, the run $p(c_a c_b^k)^\omega$ should be accepted. So it is necessary that for all k, $\mathrm{payoff}_{\mathcal{A}'_1}(c_a c_b^k) \leq 0$. Thus $\mathrm{payoff}_{\mathcal{A}'_1}(c_a) + k\,\mathrm{payoff}_{\mathcal{A}'_1}(c_b) \leq 0$, which is possible if and only if $\mathrm{payoff}_{\mathcal{A}'_1}(c_b) \leq 0$ which contradicts $\mathrm{payoff}_{\mathcal{A}'_1}(c_b) > 0$. Thus L_1 cannot be accepted by a $\mathcal{L}_\leq$ condition.

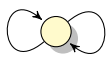
Conversely, consider the automaton



with the language $L_2 \stackrel{def}{=} \{w | \min \mathrm{Acc}(w) \leq 0\}$ of ω-words having prefixes with an arbitrarily low proportion of as. Towards contradiction, assume that $\mathcal{A}'_2$ is an automaton accepting L_2 with the $\mathcal{L}_<$ acceptance contradiction. If c_a, c_b and p are defined in $\mathcal{A}'_2$, the same way as before, we should have $\mathrm{payoff}_{\mathcal{A}'_2}(c_a) \geq 0$ and $\mathrm{payoff}_{\mathcal{A}'_2}(c_b) < 0$. For all k, the run $p(c_a c_b^k)$ should be rejected, so it is necessary that $\mathrm{payoff}_{\mathcal{A}'_2}(c_a c_b^k) \geq 0$. Thus for all k, $\mathrm{payoff}_{\mathcal{A}'_2}(c_a) + k\,\mathrm{payoff}_{\mathcal{A}'_2}(c_b) \geq 0$, which is possible if and only if $\mathrm{payoff}_{\mathcal{A}'_2}(c_b) \geq 0$ and contradicts $\mathrm{payoff}_{\mathcal{A}'_2}(c_b) < 0$. Therefore L_2 cannot be expressed by a $\mathcal{L}_<$ condition.

These counter examples can be adapted for proving that any class with strict inequality symbol is incomparable to any class with a weak inequality symbol. It remains to prove the incompatibility of $\mathcal{L}_<$ and $\mathcal{L}_>$ and that of $\mathcal{L}_\leq$ and $\mathcal{L}_\geq$.

Consider the language L_3 defined by the automaton



(denoted by $\mathcal{A}_3$) with the $\mathcal{L}_\leq$ acceptance condition: $L_3 \stackrel{def}{=} \{w | \min \mathrm{Acc}(w) \leq 0\}$, that is the set of ω-words having infinitely many prefixes with no more bs than as. Suppose, towards contradiction, that $\mathcal{A}'_3$ is an automaton defining the same language with a $\mathcal{L}_\geq$ condition. Choose two cycles c_a and c_b starting from two states q_a and q_b in a same terminal SCC of $\mathcal{A}'_3$, such that c_a is labelled by a^l and c_b is labelled by b^m for

some l and m, an initial run p going to q_b and two runs u_{ab} and u_{ba} going from q_a to q_b and from q_b to q_a, respectively. Because $pu_{ba}c_a^\omega$ should be accepted and pc_b^ω should be rejected, we must have $\text{payoff}_{\mathcal{A}'_3}(c_b) < 0$ and $\text{payoff}_{\mathcal{A}'_3}(c_a) \geq 0$. Consider $r = p\prod_{i\in\mathbb{N}} c_b^{2^i l}u_{ba}c_a^{2^i m}u_{ab}$. Then $\text{mp}_{\mathcal{A}_3}(\lambda(p\prod_{i=0}^{n} c_b^{2^i l}u_{ba}c_a^{2^i m}u_{ab}))$ converges to 0 as n goes to infinity, thus $\lambda(r) \in L_3$, and therefore r should be accepted by $\mathcal{A}'_3$ with the $\mathfrak{L}_{\geq}$ condition. But $\text{mp}_{\mathcal{A}'_3}(\lambda(p\prod_{i=0}^{n} c_b^{2^i l}u_{ba}c_a^{2^i m}u_{ab}c_b^{2^{n+1}l}))$ converges to a negative limit, thus r cannot be accepted by $\mathcal{A}'_3$ with the $\mathfrak{L}_{\geq}$ condition, leading to contradiction.

Conversely, consider the language L_4, defined with the same automaton, but with the $\mathfrak{L}_{\geq}$ condition: $L_4 \stackrel{def}{=} \{w | \min \text{Acc}(w) \geq 0\}$, the language of ω-words having finitely many prefixes with more as than bs. Suppose $\mathcal{A}_4$ is an automaton defining the same L_4 with the $\mathfrak{L}_{\leq}$ condition. We can choose two cycles c_a and c_b starting from two states q_a and q_b in a same terminal SCC of $\mathcal{A}'_4$ such that c_a is labelled by a^l and c_b is labelled by b^m for some l and m, an initial run p going to q_b and two runs u_{ab} and u_{ba} going from q_a to q_b and from q_b to q_a. Then we should have $\text{payoff}_{\mathcal{A}'_4}(c_b) \leq 0$ and $\text{payoff}_{\mathcal{A}'_4}(c_a) > 0$ (because $pu_{ba}c_a^\omega$ should be rejected and pc_b^ω should be accepted). Consider $r = p\prod_{i\in\mathbb{N}} u_{ba}c_a^{2^i m}u_{ab}c_b^{2^i l}$. Then $\text{mp}_{\mathcal{A}_3}(\lambda(p(\prod_{i=0}^{n} u_{ba}c_a^{2^i m}u_{ab}c_b^{2^i l})u_{ba}c_a^{2^{n+1}m}))$ converges to a negative limit as n goes to infinity, thus $\lambda(r) \notin L$, and hence r should be rejected by $\mathcal{A}'_4$ with the $\mathfrak{L}_{\leq}$ condition. But we show that r is actually accepted by $\mathcal{A}'_4$ with the $\mathfrak{L}_{\leq}$ condition, as $\text{mp}_{\mathcal{A}'_4}(\lambda(p\prod_{i=0}^{n} u_{ba}c_a^{2^i m}u_{ab}c_b^{2^i l}))$ converges to 0. This which contradicts the fact that $\mathcal{A}'_4$ recognizes L.

We have thus established the incomparability of $\mathfrak{L}_{\geq}$ and $\mathfrak{L}_{\leq}$. As $\mathfrak{L}_{<}$ and $\mathfrak{L}_{>}$ are the classes of the complements of languages in respectively $\mathfrak{L}_{\geq}$ and $\mathfrak{L}_{\leq}$, it also implies the incomparability of the latter pair. □

The incompatibility of threshold classes shows how arbitrary the choice of only one of them as a standard definition is. This suggests definition of a larger class including all of those acceptance conditions.

3.3.4 Mean-Payoff Languages in the Borel Hierarchy

For a topological space X, we denote by Σ_1^0 the set of open subsets by and Π_1^0 the set of closed subsets. The Borel hierarchy is defined inductively as the two sequences (Σ_α^0) and (Π_α^0), where $\Sigma_\alpha^0 = (\bigcup_{\beta<\alpha}\Pi_\beta^0)_\sigma$, and $\Pi_\alpha^0 = (\bigcup_{\beta<\alpha}\Sigma_\beta^0)_\delta$, where α and β are ordinals and $(\bullet)_\sigma$, $(\bullet)_\delta$ denote closures under countable intersections and unions, respectively.

We consider the standard topology over A^ω with the base $\{wA^\omega \colon w \in A^*\}$, i.e. a subset of A^ω is open if and only if it is a union of sets, each set consists of all possible continuations of a finite word.

Theorem 8. *The following facts hold:* $\mathfrak{L}_{\leq} \subset \Pi_2^0$, $\mathfrak{L}_{\leq} \not\subseteq \Sigma_2^0$, $\mathfrak{L}_{<} \subset \Sigma_3^0$ *and* $\mathfrak{L}_{<} \not\subseteq \Pi_3^0$.

Proof.

- Let $L \in \mathcal{L}_{\leq}$, then there exists $d \in \mathbb{N}$, a 1-payoff automaton $\mathcal{A}$ such that $L = \{w \in A^\omega | \min(Acc_{\mathcal{A}}(w)) \leq 0\}$. Therefore we can write L as

$$\begin{aligned} L &= \bigcap_{N \in \mathbb{N}} \{w \in A^\omega | \forall m \in \mathbb{N} \exists n > m \operatorname{mp}_{\mathcal{A}}(w\restriction n) < 1/N)\} \\ &= \bigcap_{N \in \mathbb{N}, m \in \mathbb{N}} \bigcup_{n > m} \{w \in A^\omega | \operatorname{mp}_{\mathcal{A}}(w\restriction n) < 1/N)\}. \end{aligned}$$

  For any N and m the condition $\operatorname{mp}_{\mathcal{A}}(w\restriction n) < 1/N)$ is independent of the suffix past the nth symbol of w and therefore the set $\{w \in A^\omega | \operatorname{mp}_{\mathcal{A}}(w\restriction n) < 1/N)\}$ is clopen. We get that $\mathcal{L}_{\leq} \in \Pi^0_2$.

- We prove $\mathcal{L}_{>} \not\subseteq \Pi^0_2$, which is the same as $\mathcal{L}_{\leq} \not\subseteq \Sigma^0_2$ because $\mathcal{L}_{>} = \operatorname{co}\mathcal{L}_{\leq}$. Let L be the set of words on alphabet $A = \{a, b\}$ having more than negligibly many b. We already demonstrated that $L \in \mathcal{L}_{>}$. Suppose $L \in \Pi^0_2$. Then $L = \bigcap_{i \in \mathbb{N}} L_i A^\omega$ for some family of languages of finite words L_i. We can assume without loss of generality that the words of L_i have all length i. For all m, the word $w_m = (\prod_{j=1}^{m-1} a^{2^j} b)(a^{2^m} b)^\omega \in L$ (as it is ultimately periodic with a period where the proportion of b is not 0). For the word $w = \prod_{j=1}^{\infty} a^{2^j} b$, it means that any prefix $w\restriction i$ of length i is in L_i. This is a contradiction, because $w \notin L$.

- For the two last items of the theorem: Chatterjee exhibited in [22] a Π^0_3-hard language in $\mathcal{L}_{\geq}$. He also established that this class is included in Π^0_3. As $\mathcal{L}_{\geq} = \operatorname{co}\mathcal{L}_{<}$, that proves what we need.

□

3.3.5 Dimensionality

In this section we analyze closure properties of mean-payoff languages defined by automata with a fixed dimension.

The following lemma shows that, for any d, the class of mean-payoff languages definable by d-payoff automata is not closed under intersection.

Lemma 8. *If d_1 and d_2 are two integers, then there exists L_1 and L_2, two mean-payoff languages of dimensions d_1 and d_2 such that L_1 and L_2 contain only convergent words and $L_1 \cap L_2$ is not definable as a dimension d mean-payoff language with $d < d_1 + d_2$.*

Proof. Let $A = \{a_1, \ldots, a_{d_1}\}$ and $B = \{b_1, \ldots, b_{d_2}\}$ be two disjoint alphabets. Let $\mathcal{A}_1$ be the one-state d_1-payoff automaton on alphabet $A \cup B$, such that the payoff of the transition (q_0, a_i, q_0) is 1 on the ith coordinate and 0 in the other coordinates and the payoff of the transition (q_0, b_i, q_0) is 0. And let $\mathcal{A}_2$ be the d_2-payoff one-state automaton defined similarly by swapping a and b.

Let L_i be the language defined on $\mathcal{A}_i$ by predicate F_i, testing equality with the singleton $\{l_i\}$, where l_i is in the simplex defined by the $d_i + 1$ different payoffs of the transitions of $\mathcal{A}_i$. In the proof of Theorem 6 we establish that the L_i are not empty.

Let $u \in (A + B)^\omega$, then u is in L_1 if and only if the proportion of a_i in every prefix tends to the ith coordinate of l_1, and it is in L_2 if and only if the proportion of b_i in every prefix tends to the ith coordinate of l_2.

Then for u to be in $L_1 \cap L_2$, it is necessary that the proportion of every symbols tends to either a coordinate of l_1, if that symbol is a a_i, or a coordinate of l_2, if it is a b_i.

Now suppose $L_1 \cap L_2$ is recognized by a d-payoff automaton with $d < d_1 + d_2$. Choose one terminal SCC of $\mathcal{A}$ and consider for every symbol a of the alphabet a cycle labeled by a word in a^*, starting at some state q_a. Let also be p an initial run going to q_a and for every pair of symbols a, b a path u_{ab} going from q_a to q_b.

Only looking at the runs in the language $p\{u_{a_1 a} c_a^* u_{a a_1} | a \in A \cup B\}^\omega$, it is possible to converge to any proportion of the symbols of $A \cup B$, and thus have runs whose labeling word is in L. But as the payoffs are in dimension d, and the number of symbols is $d_1 + d_2 > d$, that language also contains runs converging to different symbol proportions but still having the same mean-payoff limit. Those runs are accepted by $\mathcal{A}$ but are not labeled by a word in L. □

Next, we prove that the intersection of two languages of dimensions d_1 and d_2 is a language of dimension $d_1 + d_2$. This will be proved constructively, by showing that the intersection language is the language defined on the product automaton with the "product" condition. Before going to the statement of the lemma, we need to define what those products are.

Definition 19. *If F_1 and F_2 are predicates on $2^{\mathbb{R}^{d_1}}$ and $2^{\mathbb{R}^{d_2}}$, we denote by $F_1 \Cap F_2$ the predicate on $2^{\mathbb{R}^{d_1+d_2}}$ which is true for $X \subseteq \mathbb{R}^{d_1+d_2}$ if and only if $F_1(p_1(X))$ and $F_2(p_2(X))$, where p_1 is the projection on the d_1 first coordinates and p_2 on the d_2 last ones.*

Definition 20 (Weighted automata product)**.** *If $\mathcal{A}_1 = \langle A, Q_1, q_0^1, \delta_1, w_1 \rangle$ is a d_1-payoff automaton and $\mathcal{A}_2 = \langle A, Q_2, q_0^2, \delta_2, w_2 \rangle$, a d_2-payoff automaton, then we define $\mathcal{A}_1 \otimes \mathcal{A}_2 = \langle A, Q_1 \times Q_2, (q_0^1, q_0^2), \delta_{1\otimes 2}, w_{1\otimes 2} \rangle$, the product of $\mathcal{A}_1$ and $\mathcal{A}_2$, a $(d_1 + d_2)$-payoff automaton such that*

- $\delta_{1\otimes 2} = \{((q_1, q_2), a, (q_1', q_2')) | (q_1, a, q_1') \in \delta_1 \wedge (q_2, a, q_2') \in \delta_2 \wedge a \in A\}$,
- $w_{1\otimes 2} \colon \delta_{1\otimes 2} \to \mathbb{R}^{d_1+d_2}$ *is such that*
  if $w_1(q_1, a, q_1') = (x_1, \dots x_{d_1})$ and $w_2(q_2, a, q_2') = (y_1, \dots y_{d_2})$,
  then $w((q_1, q_2), a, (q_1', q_2')) = (x_1, \dots x_{d_1}, y_1, \dots y_{d_2})$.

But before we state the theorem, we need the following lemma:

Lemma 9. *If r is a run of a d-payoff automaton $\mathcal{A}$ and p is a projection from $\mathbb{R}^d$ to $\mathbb{R}^{d'}$, with $d' < d$, then* $\mathrm{Acc}(p(\mathrm{mp}_{\mathcal{A}}(r\restriction n))) = p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r\restriction n)))$

Proof. Let $x' \in \mathrm{Acc}(p(\mathrm{mp}_{\mathcal{A}}(r \restriction n)))$. For any $i \in \mathbb{N}$, $p(\mathrm{mp}_{\mathcal{A}}(r \restriction n))$ eventually comes into a distance $1/i$ from x', for some index n_i. For $j > i$ $\mathrm{mp}_{\mathcal{A}}(r \restriction n_j)$ remains in $B(x', 1/i) \times K$ (where K is a compact of $\mathbb{R}^{d-d'}$), as this product is compact, it has at least one accumulation point. Thus the distance from x' to $p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r \restriction n)))$ is 0. But $\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r \restriction n))$ is a closed set and p, being a projection, is continuous, so $p(\mathrm{Acc}(\mathrm{mp}(r \restriction n)))$ is closed too, which means $x' \in p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r \restriction n)))$, and so $\mathrm{Acc}(p(\mathrm{mp}(r \restriction n))) \subseteq p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r \restriction n)))$.

Conversely, if $x' \in p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r \restriction n)))$ a sub-sequence $\mathrm{mp}_{\mathcal{A}}(r \restriction n_i)$ converges to a x such that $x' = p(x)$, and thus $p(\mathrm{mp}_{\mathcal{A}}(r \restriction n_i))$ converges to x', which means $x' \in \mathrm{Acc}(p(\mathrm{mp}(r \restriction n)))$. We conclude that $\mathrm{Acc}(p(\mathrm{mp}_{\mathcal{A}}(r \restriction n))) = p(\mathrm{Acc}(\mathrm{mp}_{\mathcal{A}}(r \restriction n)))$. □

Now we have all the needed tools, we can characterize the intersection of two mean-payoff languages as another mean-payoff language defined on an automaton whose dimension is known.

Proposition 3. *For any two d_1-payoff and d_2-payoff automata $\mathcal{A}_1$ and $\mathcal{A}_2$ and any two predicates F_1 and F_2 on respectively $2^{\mathbb{R}^{d_1}}$ and $2^{\mathbb{R}^{d_2}}$, the following equality holds: $L(\mathcal{A}_1, F_1) \cap L(\mathcal{A}_2, F_2) = L(\mathcal{A}_1 \otimes \mathcal{A}_2, F_1 \Cap F_2)$.*

Proof. Suppose $u \in A^\omega$, then the sequence of mean-payoffs of run r of u in $\mathcal{A}_1 \otimes \mathcal{A}_2$ are the projections by p_1 and p_2 of the sequence of mean-payoffs of some runs r_1 and r_2 in $\mathcal{A}_1$ and r_2 in $\mathcal{A}_2$ whose labeling is u. And conversely, if u has runs r_1 and r_2 in $\mathcal{A}_1$ and r_2 in $\mathcal{A}_2$, then it has a run r in $\mathcal{A}_1 \otimes \mathcal{A}_2$ whose sequence of mean-payoffs projects by p_1 and p_2 onto those of r_1 and r_2.

If r, r_1, and r_2 are such runs (the payoffs of r projecting on those of r_1 and r_2), then using lemma 9, we find that $\mathrm{Acc}_{\mathcal{A}}(r_1) = \mathrm{Acc}(p_1(\mathrm{mp}(r \restriction n))) = p_1(\mathrm{Acc}(\mathrm{mp}(r \restriction n)))$ and that $\mathrm{Acc}_{\mathcal{A}}(r_2) = \mathrm{Acc}(p_2(\mathrm{mp}(r \restriction n))) = p_2(\mathrm{Acc}(\mathrm{mp}(r \restriction n)))$.

But by definition $(F_1 \Cap F_2)(\mathrm{Acc}(\mathrm{mp}(r \restriction n)))$ holds iff $F_1(p_1(\mathrm{Acc}(\mathrm{mp}(r \restriction n))))$ and $F_2(p_2(\mathrm{Acc}(\mathrm{mp}(r \restriction n))))$ hold, thus it holds iff $F_1(\mathrm{Acc}_{\mathcal{A}_1}(r_1))$ and $F_2(\mathrm{Acc}_{\mathcal{A}_2}(r_2))$ hold.

From that we deduce that a word is in $L(\mathcal{A} \otimes \mathcal{A}, F_1 \Cap F_2)$ if and only if it is both in $L(\mathcal{A}_1, F_1)$ and $L(\mathcal{A}_2, F_2)$. □

3.4 An Analyzable Class of Mean-Payoff Languages

3.4.1 Multi-Threshold Mean-Payoff Languages

As a candidate for a class of mean-payoff languages that is closed under complementation and includes all the expected standard mean-payoff language classes, we propose the following definition.

Definition 21. *A language L is a multi-threshold mean-payoff language (denoted by $L \in \mathfrak{L}_{mt}$) if it is the mean-payoff language defined on some d-payoff automaton $\mathcal{A}$, with a predicate F such that $F(S)$ is a Boolean combination of threshold conditions on $p_i(S)$ where p_i is the projection along the ith coordinate.*

Example 3. *Consider the automaton given in Example 1 and the multi-threshold mean-payoff language*

$$L = \{w| \quad \min p_1(\mathrm{Acc}(w)) > .1 \wedge \max p_1(\mathrm{Acc}(w)) < .9 \\ \wedge \min p_2(\mathrm{Acc}(w)) > .1 \wedge \max p_2(\mathrm{Acc}(w)) < .9 \quad \}.$$

For the word w, defined in Example 1, the set of accumulation points is shown to be a triangle that is contained in the box $\{x|.1 < p_1(x) < .9 \wedge .1 < p_2(x) < .9\}$ and therefore $w \in L$.

Geometrically, multi-threshold acceptance conditions can be visualized as specifying constraints on the maximal and minimal projection of $\mathrm{Acc}(w)$ on the axes. Since we can extend the payoff vectors by adding a coordinate whose values are a linear combination of the other coordinates, also threshold constraints involving minimal and maximal elements of the projection of $\mathrm{Acc}(w)$ on other lines are expressible, as shown in the following example.

Example 4. *Assume that, with the automaton given in Example 1, we want to accept the words w such that $\mathrm{Acc}(w)$ is contained in the triangle $(.2, .2) - (.8, .2) - (.2, .8)$. We can do so by extending the dimension of the payoffs and renaming $(0, 0) \mapsto (0, 0, 0)$, $(1, 0) \mapsto (1, 0, 1)$, and $(1, 1) \mapsto (1, 1, 2)$. Namely, by adding a coordinate whose value is the sum of the other two coordinates. Then, $L = \{w| \min p_1(\mathrm{Acc}(w)) > .2 \wedge \min p_2(\mathrm{Acc}(w)) > .2 \wedge \max p_3(\mathrm{Acc}(w)) < 1\}$ is the wanted language.*

3.4.2 Closure under Boolean operations

We prove here that $\mathcal{L}_{mt}$ is in fact the Boolean closure of $\mathcal{L}_{\substack{\leq \\ >}} \stackrel{def}{=} \mathcal{L}_< \cup \mathcal{L}_\leq \cup \mathcal{L}_> \cup \mathcal{L}_\geq$.

Theorem 9. *$\mathcal{L}_{mt}$ is closed under Boolean operations and any language in $\mathcal{L}_{mt}$ is a Boolean combination of languages in $\mathcal{L}_{\substack{\leq \\ >}}$.*

Proof. Closure by complementation: let L be a $\mathcal{L}_{mt}$ language, defined on some automaton $\mathcal{A}$ by a predicate P. $w \in L$ iff $P(\mathrm{Acc}_\mathcal{A}(w))$. So $w \in L^c$ iff $w \notin L$, that is iff $\neg P(\mathrm{Acc}_\mathcal{A}(w))$. But $\neg P$ is also a Boolean combination of threshold conditions, thus L^c is a $\mathcal{L}_{mt}$ language.

Closure by intersection: let L_1 and L_2 be two $\mathcal{L}_{mt}$ languages defined respectively on the automata $\mathcal{A}$ and $\mathcal{A}$ by the predicates P_1 and P_2. Then $L_1 \cap L_2 = L(\mathcal{A} \otimes \mathcal{A}, P_1 \Cap P_2)$ (Theorem 3). It is easy to see that $P_1 \Cap P_2$ is still a Boolean combination of thresholds, and thus $L(\mathcal{A} \otimes \mathcal{A}, P_1 \Cap P_2)$ is in $\mathcal{L}_{mt}$.

The other Boolean operations can be written as a combination of complementation and intersection.

Now we show, by induction on height of the formula of a predicate, that any $\mathcal{L}_{mt}$ language is a Boolean combination of $\mathcal{L}_{\substack{\leq \\ >}}$ languages.

We can, without loss of generality, suppose that every threshold concerns a different coordinate (if a coordinate has several thresholds, we can duplicate that coordinate,

keeping the same values, and the language will remain the same). We can also assume that the predicate is written only with combinations of conjunctions and negations of thresholds.

- If the height is 0, that means that the condition is only one threshold on a multi-payoff automaton. The recognized language is the same as that of the automaton projected on the tested coordinate, so it is in $\mathcal{L}_{\substack{\leq \\ >}}$.
- If the predicate is $\neg P$, then the recognized language is the complement of $L(\mathcal{A}, P)$, which is a Boolean combination of $\mathcal{L}_{mt}$ languages of lesser height.
- If the predicate is $P = P_1 \wedge P_2$, let us call $\mathcal{A}_i$, a copy of $\mathcal{A}$ whose payoffs are projected on the subspace tested in P_i. Then $\mathcal{A}$ is isomorphic to $\mathcal{A}_1 \otimes \mathcal{A}_2$. Furthermore, as the set of coordinates that are tested in P_1 and P_2 are disjoint, their exists some P_1' and P_2' with the same heights as P_1 and P_2, such that $P = P_1' \Cap P_2'$. Thus $L = L(\mathcal{A}_1, P_1') \cap L(\mathcal{A}_2, P_2')$ (Theorem 3), which is a Boolean combination of $\mathcal{L}_{mt}$ languages of lesser height.

□

3.4.3 Decidability

Theorem 10. *The emptiness of a language of $\mathcal{L}_{mt}$ is decidable.*

Proof. We can assume the acceptance predicate is written in disjunctive normal form (if not, we can find an equivalent DNF formula). Then we can see that a run is accepted whenever its set of accumulation points satisfies at least one of the disjuncts, and in a disjunct, every literal has to be satisfied. If we know how to check if a literal is satisfied, then this provides an obvious decision algorithm for one run.

Then it is easy to see that there are two types of literals. Some require that there must exist an accumulation point whose tested coordinate is greater or smaller than the threshold, we call those existential literals. The other literals require that every accumulation point should have the tested coordinate above or below the threshold, those we call universal literals.

For checking the emptiness of $L(\mathcal{A}, F)$, we propose algorithm 2.

If this algorithm returns that $L(\mathcal{A}, F)$ is not empty, it means that I is a convex polyhedron included in C and intersecting with every existential literal of some disjunct of F and that we can construct I' which is connected, closed, included in I, and intersects with every existential literal (take for instance the convex hull of a family consisting in one point in every intersection of I with an existential literal). We can see that $F(I')$ holds. Then Theorem 6 says there exist a run r such that $\mathrm{Acc}_{\mathcal{A}}(r) = I'$, and the word whose run is r is in $L(\mathcal{A}, F)$.

If this algorithm returns that $L(\mathcal{A}, F)$ is empty, then for every reachable SCC C, if you choose a closed connected subset D of $P(C)$ then, for every disjunct, either D

Algorithm 2 Multi-threshold mean-payoff language emptiness

```
for all disjunct δ of F do
    for all reachable SCC C of A do
        P(C) := convex hull of the payoffs of transitions in C
        I := intersection of P(C) with every universal literal of δ
        if I = ∅ then consider next disjunct
        for all existential literal e of δ do
            if I ∩ e = ∅ then consider next disjunct
        return "L(A, F) ≠ ∅"
return "L(A, F) = ∅"
```

is not completely included in some universal literal or D does not intersect with some existential literal. In both case, D does not make the disjunct true. But Theorem 5 states that sets of accumulation points have to be closed, connected, and in $P(C)$ for some C. Thus F holds for no set of accumulation points of any run of $\mathcal{A}$, which implies that $L(\mathcal{A}, F)$ is empty. □

3.5 Summary and Future Directions

We proposed a definition of ω-languages using Boolean combinations of threshold predicates over mean-payoffs. This type of specifications allows to express requirements concerning averages such as "no more than 10% of the messages are lost" or "the number of messages lost is negligible". The latter is not expressible by ω-regular requirements. We showed that if closure under intersection is needed, multi-dimensional payoffs have to be considered. For runs of d-payoff automata, we studied acceptance conditions that examine the set of *accumulation points* and characterized those sets as all closed, bounded and connected subsets of $\mathbb{R}^d$.

The class of *multi-threshold mean-payoff languages* was proposed, using acceptance conditions that are Boolean combinations of inequalities comparing the minimal or maximal accumulation point along some coordinate with a constant threshold. We studied expressiveness, closure properties, analyzability, and Borel complexity.

Possible directions for future include extension to non-deterministic automata, and the study of multi-mean-payoff games.

Chapter 4

Volume and entropy of regular timed languages

For timed languages, we define size measures: volume for languages with a fixed finite number of events, and entropy (growth rate) as asymptotic measure for an unbounded number of events. These measures can be used for quantitative comparison of languages, and the entropy can be viewed as information contents of a timed language. For languages accepted by deterministic timed automata, we give exact formulas for volumes. Next, we characterize the entropy, using methods of functional analysis, as a logarithm of the leading eigenvalue (spectral radius) of a positive integral operator. We devise several methods to compute the entropy: a symbolical one for so-called "1 $\frac{1}{2}$-clock" automata, and two numerical ones: one using techniques of functional analysis, another based on discretization. We give an information-theoretic interpretation of the entropy in terms of Kolmogorov complexity.

4.1 Introduction

Since early 90s, timed automata and timed languages are extensively used for modelling and verification of real-time systems, and thoroughly explored from a theoretical standpoint. However, two important, and closely related, aspects have never been addressed: quantitative analysis of the size of these languages and of information content of timed words. In this chapter, we formalize and solve these problems for a large subclass of timed automata.

Recall that a timed word describes a behaviour of a system, taking into account delays between events. For example, $2a3.11b$ means that an event a happened 2 time units after the system start, and b happened 3.11 time units after a. A timed language, which is just a set of timed words, may represent all such potential behaviours. Our aim is to measure the size of such a language. For a fixed number n of events, we can consider the language as a subset of $\Sigma^n \times \mathbb{R}^n$ (that is of several copies of the space $\mathbb{R}^n$). A natural measure in this case is just Euclidean volume V_n of this subset. When

the number of events is not fixed, we can still consider for each n all the timed words with n events belonging to the language and their volume V_n. It turns out that in most cases V_n asymptotically behaves as $2^{n\mathcal{H}}$ for some constant $\mathcal{H}$ that we call entropy of the language.

The information-theoretic meaning of $\mathcal{H}$ can be stated as follows: for a small ε, if the delays are measured with a finite precision ε, then using the words of the language L with entropy $\mathcal{H}$ one can transmit $\mathcal{H} + \log(1/\varepsilon)$ bits of information per event (see Thms. 7-8 below for a formalization in terms of Kolmogorov complexity).

There can be several potential applications of these notions:

- The most direct one is capacity estimation for an information transmission channel or for a time-based information flow.
- When one overapproximates a timed language L_1 by a simpler timed language L_2 (using, for example, some abstractions as in [12]), it is important to assess the quality of the approximation. Comparison of entropies of L_1 and L_2 provides such an assessment.
- In model-checking of timed systems, it is often interesting to know the size of the set of all behaviours violating a property or of a subset of those presented as a counter-example by a verification tool.

In this chapter, we explore, and partly solve the following problems: given a prefix-closed timed language accepted by a timed automaton, find the volume V_n of the set of accepted words of a given length n and the entropy $\mathcal{H}$ of the whole language.

Related Work.

Our problems and techniques are inspired by works concerning the entropy of finite-state languages (cf. [41], where entropy is studied in the context of dynamical systems, yielding precise results for sofic shifts, in other words finite-state automata). There the cardinality of the set L_n of all elements of length n of a prefix-closed regular language also behaves as $2^{n\mathcal{H}}$ for some entropy $\mathcal{H}$. This entropy can be found as logarithm of the spectral radius of adjacency matrix of reachable states of $\mathcal{A}$.[1] The main technical tool used to compute the entropy of finite automata is the Perron-Frobenius theory for positive matrices, and, in this chapter, in a first approach we will use its extensions to infinite-dimensional operators [38]. In a second approach, we also propose to reduce our problem by discretization to entropy computation for some discrete automata.

In [18, 49] probabilities of some timed languages and densities in the clock space are computed. Our formulae for fixed-length volumes can be seen as specialization of these results to uniform measures. As for unbounded languages, they use stringent condition of full simultaneous reset of all the clocks at most every k steps, and under such a condition, they provide a finite stochastic class graph that allows computing various

[1] This holds also for automata with multiplicities, see [41].

interesting probabilities. We use a much weaker hypothesis (every clock to be reset at most every D steps, but these resets need not be simultaneous), and we obtain only the entropy.

In [13] probabilities of LTL properties of one-clock timed automata (over infinite timed words) are computed using Markov chains techniques. It would be interesting to try to adapt our methods to this kind of problems.

Last, our studies of Kolmogorov complexity of rational elements of timed languages, relating this complexity to the entropy of the language, remind of earlier works on complexity of rational approximations of continuous functions [11, 48], and those relating complexity of trajectories to the entropy of dynamical systems [17, 48].

Chapter Organization

This chapter is organized as follows. In Sect. 4.2 we define volumes of fixed-length timed languages and entropy of unbounded-length timed languages. We identify a subclass of deterministic timed automata, whose volumes and entropy are considered in the rest of the chapter, and a normal form for such automata. Finally, we provide an algorithm for computing the volumes of languages of such automata. In Sect. 4.3 we define a functional space associated to a timed automaton and a positive operator on this space. We rephrase the formulas for the volume in terms of this operator. Next, we state the main result of the chapter: a characterization of the entropy as the logarithm of the spectral radius of this operator. Such a characterization could seem too abstract but later on, in sections 4.4-4.5 we give three practical procedures for approximate computing this spectral radius. First, we show how to solve the eigenvector equation symbolically in case of timed automata with $1\frac{1}{2}$ clocks defined below. Next, for general timed automata we apply a "standard" iterative procedure from [38] and thus obtain an upper and a lower bound for the spectral radius/entropy. These bounds become tighter as we make more iterations. Last, in Sect. 4.5, also for general timed automata, we devise a procedure that provides upper and lower bounds of the entropy by discretization of the timed automaton. In the same section, and using the same method, we give an interpretation of the entropy of timed regular languages in terms of Kolmogorov complexity. We conclude the chapter by some final remarks in Sect. 4.7. Throughout the chapter, the concepts and the techniques are illustrated by several running examples.

4.2 Problem Statement

4.2.1 Geometry, Volume and Entropy of Timed Languages

We recall that a *timed word* of length n over an alphabet Σ is a sequence $w = t_1 a_1 t_2 \dots t_n a_n$, where $a_i \in \Sigma, t_i \in \mathbb{R}$ and $0 \leq t_i$ (notice that in this chapter we rule out timed words ending by a time delay), where t_i represents the delay between the events a_{i-1} and a_i. We introduce, for a timed word w of length n, its *untiming*

$\eta(w) = a_1, \ldots, a_n \in \Sigma^n$ (which is just a discrete word), and its *timing* which is a point $\theta(w) = (t_1, \ldots, t_n)$ in $\mathbb{R}^n$. A *timed language* L is a set of timed words. For a fixed n, we define the *n-volume* of L as follows:

$$V_n(L) = \sum_{v \in \Sigma^n} \mathrm{Vol}\{\theta(w) \mid w \in L, \eta(w) = v\},$$

where Vol stands for the standard Euclidean volume in $\mathbb{R}^n$. In other words, we sum up over all the possible untimings v of length n the volumes of the corresponding sets of delays in $\mathbb{R}^n$. In case of regular timed languages, these sets are polyhedral, and hence their volumes (finite or infinite) are well-defined.

We associate with every timed language a sequence of n-volumes V_n. We will show in Sect. 4.2.5 that, for languages of deterministic timed automata, V_n is a computable sequence of rational numbers. However, we would like to find a unique real number characterizing the asymptotic behaviour of V_n as $n \to \infty$. Typically, V_n depends approximately exponentially on n. We define the entropy of a language as the rate of this dependence.

Formally, for a timed language L we define its *entropy* as follows[2] (all logarithms in the chapter are base 2):

$$\mathcal{H}(L) = \limsup_{n \to \infty} \frac{\log V_n}{n}.$$

Remark 1. *Many authors consider a slightly different kind of timed words: sequences $w = (a_1, d_1), \ldots, (a_n, d_n)$, where $a_i \in \Sigma, d_i \in \mathbb{R}$ and $0 \le d_1 \le \cdots \le d_n$, with d_i representing the date of the event a_i. This definition is in fact isomorphic to ours by a change of variables: $t_1 = d_1$ and $t_i = d_i - d_{i-1}$ for $i = 2..n$. It is important for us that this change of variables preserves the n-volume, since it is linear and its matrix has determinant 1. Therefore, choosing date (d_i) or delay (t_i) representation has no influence on language volumes (and entropy). As both notations are equivalent, we prefer using the same convention as in the first chapter of the thesis.*

4.2.2 Three Examples

To illustrate the problem consider the languages recognized by three timed automata on Fig. 4.1. Two of them can be analysed directly, using definitions and common sense. The third one resists naive analysis, it will be used to illustrate more advanced methods throughout the chapter.

Rectangles.

Consider the timed language defined by the expression

$$L_1 = ([2;4]a + [3;10]b)^*,$$

[2] In fact, due to Assumption A2 below, the languages we consider in the chapter are prefix-closed, and lim sup is in fact a lim. This will be stated formally in Cor. 1.

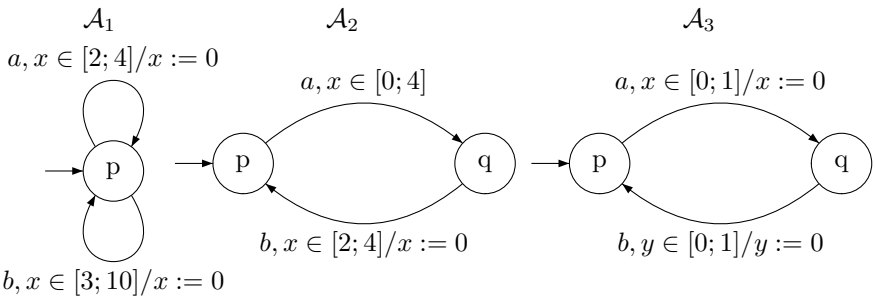


Figure 4.1: Three simple timed automata $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$

recognized by $\mathcal{A}_1$ of Fig. 4.1.

For a given untiming $w \in \{a, b\}^n$ containing k letters a and $n-k$ letters b, the set of possible timings is a rectangle in $\mathbb{R}^n$ of a volume $2^k 7^{n-k}$ (notice that there are C_n^k such untimings). Summing up all the volumes, we obtain

$$V_n(L_1) = \sum_{k=0}^{n} C_n^k 2^k 7^{n-k} = (2+7)^n = 9^n,$$

and the entropy $\mathcal{H}(L_1) = \log 9 \approx 3.17$.

A Product of Trapezia.

Consider the language defined by the automaton $\mathcal{A}_2$ on Fig. 4.1, that is containing words of the form $t_1 a s_1 b t_2 a s_2 b \dots t_k a s_k b$ such that $2 \le t_i + s_i \le 4$. Since we want prefix-closed languages, the last $s_k b$ can be omitted.

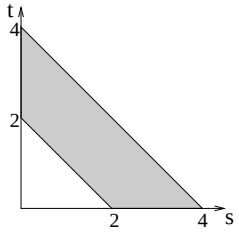


Figure 4.2: Timings (t_i, s_i) for $\mathcal{A}_2$.

For an even $n = 2k$ the only possible untiming is $(ab)^k$. The set of timings in $\mathbb{R}^{2k}$ is a Cartesian product of k trapezia $2 \le t_i + s_i \le 4$. The surface of each trapezium equals $S = 4^2/2 - 2^2/2 = 6$, and the volume $V_{2k}(L_2) = 6^k$. For an odd $n = 2k+1$ the same product of trapezia is combined with an interval $0 \le t_{k+1} \le 4$, hence the volume is $V_{2k+1}(L_2) = 6^k \cdot 4$. Thus the entropy $\mathcal{H}(L_2) = \log 6/2 \approx 1.29$.

Our Favourite Example.

The language recognized by the automaton $\mathcal{A}_3$ on Fig. 4.1 contains the words of the form $t_1 a t_2 b t_3 a t_4 b \dots$ with $t_i + t_{i+1} \in [0;1]$. Notice that the automaton has two clocks that are never reset together. The geometric form of possible untimings in $\mathbb{R}^n$ is defined by overlapping constraints $t_i + t_{i+1} \in [0;1]$.

It is not so evident how to compute the volume of this polyhedron. A systematic method is described below in Sect. 4.2.5. An *ad hoc* solution would be to integrate 1 over the polyhedron, and to rewrite this multiple integral as an iterated one. The resulting formula for the volume is

$$V_n(L_3) = \int_0^1 dt_1 \int_0^{1-t_1} dt_2 \int_0^{1-t_2} dt_3 \dots \int_0^{1-t_{n-1}} dt_n.$$

This gives the sequence of volumes:

$$1; \frac{1}{2}; \frac{1}{3}; \frac{5}{24}; \frac{2}{15}; \frac{61}{720}; \frac{17}{315}; \frac{277}{8064}; \dots$$

In the sequel, we will also compute the entropy of this language.

4.2.3 Subclasses of Timed Automata

In the rest of the chapter, we compute volumes and entropy for regular timed languages recognized by some subclasses of timed automata (TA). We recall here briefly what a TA and its language are, but the reader is advised to read [3] for details.

Definition 1. *A timed automaton is a tuple $\mathcal{A} = (Q, \Sigma, C, \Delta, q_0)$ where*

- *Q is a finite set of control locations or discrete states,*
- *Σ is a finite set of symbols, the alphabet,*
- *C is a finite set of clocks,*
- *$\Delta \subseteq Q \times \Sigma \times G \times R \times Q$ is the set of discrete transitions, where*
  - *G is the set of guards, a guard being a condition on clock values i.e. a subset of $\mathbb{R}^C$,*
  - *R is the set of resets $\mathfrak{r} \in \mathbb{R}^C \to \mathbb{R}^C$, $\mathfrak{r}(x_1, \dots, x_{|C|}) = (y_1, \dots, y_{|C|})$ where y_i is either 0 if i is a clock reset by $\mathfrak{r}$, or x_i.*
- *and $q_0 \in Q$ is the initial location.*

We do not need to specify accepting states due to A2 below, neither we need any invariants.

A generic state of $\mathcal{A}$ is a pair $(q, \mathbf{x})$ of a control location and a vector of clock values in $\mathbb{R}^C$. A generic element of Δ is written as $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q')$ meaning a transition from q to q' with label a, guard $\mathfrak{g}$ and reset $\mathfrak{r}$.

A state $(q, \mathbf{x})$ can be transformed into another state $(q', \mathbf{x}')$ by either

- a timed transition: letting τ units of time pass, such that $q' = q$ and $\mathbf{x}' = \mathbf{x} + \tau$ (τ is added to all the clock values),
- a discrete transition $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q')$, only possible if $\mathbf{x} \in \mathfrak{g}$, and implying that $\mathbf{x}' = \mathfrak{r}(\mathbf{x})$.

A run of the automaton $\mathcal{A}$ is a sequence of transitions starting from q_0. The language of the automaton $\mathcal{A}$ is the set of sequence of the labels of the its runs, where the label of a timed transition is its duration.

Several combinations of the following Assumptions will be used in the sequel:

A1. The automaton $\mathcal{A}$ is deterministic[3].

A2. All its states are accepting.

A3. Guards are rectangular (i.e. conjunctions of constraints $L_i \leq x_i \leq U_i$, strict inequalities are also allowed). Every guard upper bounds at least one clock.

A4. There exists a $D \in \mathbb{N}$ such that on every run segment of D transitions, every clock is reset at least once.

A5. There is no punctual guards, that is in any guard $L_i < U_i$.

We call an automaton *nice* if it satisfies $A1 - A4$.

Below we motivate and justify these choices:

A1: Most of known techniques to compute entropy of untimed regular languages work on deterministic automata. Indeed, these techniques count paths in the automaton, and only in the deterministic case their number coincides with the number of accepted words. The same is true for volumes in timed automata. It can be shown that any TA satisfying A4 can be determinized.[4]

A2: Prefix-closed languages are natural in the entropy context, and somewhat easier to study. These languages constitute the natural model for the set of behaviours of causal systems.

A3: If a guard of a feasible transition is infinite, the volume becomes infinite. We conclude that A3 is unavoidable and almost not restrictive.

A4: We use this variant of non-Zenoness condition several times in our proofs and constructions. As the automaton of Fig. 4.3 shows, if we omit this assumption some anomalies can occur.

The language of this automaton is

$$L = \{t_1 a \dots t_n a \mid 0 \leq \sum t_i \leq 1\},$$

[3]That is any two transitions with the same source and the same label have their guards disjoint.

[4]We thank R. Lanotte for pointing this to us.

Figure 4.3: An automaton without resets

and V_n is the volume of an n-dimensional simplex defined by the constraints $0 \leq \sum t_i \leq 1$, and $0 \leq t_i$. Hence $V_n = 1/n!$ which decreases faster than any exponent, which is too fine to be distinguished by our methods. Assumption A4 rules out such anomalies.

This assumption is also the most difficult to check. A possible way would be to explore all simple cycles in the region graph and to check that all of those reset every clock.

A5: While assumptions A1-A4 can be restrictive, we always can remove the transitions with punctual guards from any automaton, without changing the volumes V_n. Hence, A5 is not restrictive at all, as far as volumes are considered. In Sect. 4.6 we do not make this assumption.

4.2.4 Preprocessing Timed Automata

In order to compute volumes V_n and entropy $\mathcal{H}$ of the language of a nice TA, we first transform this automaton into a normal form, which can be considered as a (timed) variant of the region graph.

Although the reader can find more details about the region equivalence in [3], here we give a quick definition of it for a TA satisfying A3. We denote by M the highest finite constant against which a clock can be tested in a guard of the TA. Two clock vectors in $\mathbb{R}^C$, $(x_1, \ldots, x_{|C|})$ and $(y_1, \ldots, y_{|C|})$, are *region-equivalent* if

- for all i and j in C, $frac(x_i) \leq frac(x_j)$ iff $frac(y_i) \leq frac(y_j)$,
- for all $i \in C$, $frac(x_i) = 0$ iff $frac(y_i) = 0$,
- and for all $i \in C$, either $\lfloor x_i \rfloor = \lfloor y_i \rfloor$ or x_i and y_i are both greater than M,

where $frac(x)$ stands for the fractional part of the real number x, $\lfloor x \rfloor$ for its integer part, and c_i for the highest constant. An equivalence class of this relation is called a *clock region*, and the quotient of the TA by this equivalence is called *region graph*.

We say that a TA $\mathcal{A} = (Q, \Sigma, C, \delta, q_0)$ is in a *region-split form* if A1, A2, A4 and the following properties hold:

B1. Each location and each transition of $\mathcal{A}$ is visited by some run starting at $(q_0, 0)$.

B2. For every location $q \in Q$ a unique clock region $\mathbf{r}_q$ (called its *entry region*) exists, such that the set of clock values with which q is entered is exactly $\mathbf{r}_q$. For the initial location q_0, its entry region is the singleton $\{0\}$.

B3. The guard $\mathfrak{g}$ of every transition $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q') \in \Delta$ is just one clock region.

Notice, that B2 and B3 imply that $\mathfrak{r}(\mathfrak{g}) = \mathbf{r}_{q'}$ for every δ.

Proposition 4. *Given a nice TA $\mathcal{A}$, a region-split TA $\mathcal{A}'$ accepting the same language can be constructed* [5].

Proof sketch. Let $\mathcal{A} = (Q, \Sigma, C, \Delta, q_0)$ be a nice TA and let **Reg** be the set of its regions. The region-split automaton $\mathcal{A}' = (Q', \Sigma, C, \Delta', q_0')$ can be constructed as follows:

1. Split every state q into substates corresponding to all possible entry regions. Formally, just take $Q' = Q \times \mathbf{Reg}$.

2. Split every transition from q to q' according to two clock regions: one for the clock values when q is entered, another for clock values when q is left. Formally, for every $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q')$ of $\mathcal{A}$, and every two clock regions $\mathbf{r}$ and $\mathbf{r}'$ such that $\mathbf{r}'$ is reachable from $\mathbf{r}$ by time progress, and $\mathbf{r}' \subset \mathfrak{g}$, we define a new transition of $\mathcal{A}'$

$$\delta'_{\mathbf{r}\mathbf{r}'} = ((q, \mathbf{r}), a, \mathbf{r}', \mathfrak{r}, (q', \mathfrak{r}(\mathbf{r}'))) .$$

3. Take as initial state $q_0' = (q_0, \{0\})$.

4. Remove all the states and transitions not reachable from the initial state.

It can be shown that $\mathcal{A}$ and $\mathcal{A}'$ recognize the same language.

□

We could work with the region-split automaton, but it has too many useless (degenerate) states and transitions, which do not contribute to the volume and the entropy of the language. This justifies the following definition: we say that a region-split TA is *fleshy* if the following holds:

B4. For every transition δ its guard $\mathfrak{g}$ has no constraints of the form $x = c$ in its definition.

Proposition 5. *Given a region-split TA $\mathcal{A}$ accepting a language L, a fleshy region-split nice TA $\mathcal{A}'$ accepting a language $L' \subset L$ with $V_n(L') = V_n(L)$ and $\mathcal{H}(L') = \mathcal{H}(L)$ can be constructed.*

Proof sketch. The construction is straightforward:

1. Remove all non-fleshy transitions.

2. Remove all the states and transitions that became unreachable.

[5] Notice that due to A3 all the guards of original automaton are bounded w.r.t. some clock. Hence, the same holds for smaller (one-region) guards of $\mathcal{A}'$, that is the infinite region $[M; \infty)^{|C|}$ never occurs as a guard.

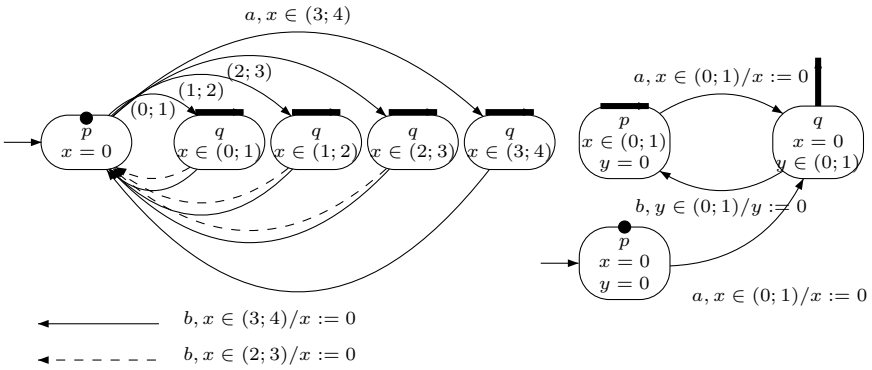


Figure 4.4: Fleshy region-split forms of automata $\mathcal{A}_2$ and $\mathcal{A}_3$ from Fig. 4.1. An entry region is drawn at each location.

Inclusion $L' \subset L$ is immediate. Every path in $\mathcal{A}$ (of length n) involving a non-fleshy (punctual) transition corresponds to the set of timings in $\mathbb{R}^n$ which is degenerate (its dimension is smaller than n), hence it does not contribute to V_n. □

From now on, we suppose w.l.o.g. that the automaton $\mathcal{A}$ is in a fleshy region-split form (see Fig. 4.4).

4.2.5 Computing Volumes

Given a timed automaton $\mathcal{A}$ satisfying A1-A3, we want to compute n-volumes V_n of its language. In order to obtain recurrent equations on these volumes, we need to take into account all possible initial locations and clock configurations. For every state $(q, \mathbf{x})$, let $L(q, \mathbf{x})$ be the set of all the timed words corresponding to the runs of the automaton starting at this state, let $L_n(q, \mathbf{x})$ be its sublanguage consisting of its words of length n, and $v_n(q, \mathbf{x})$ the volume of this sublanguage. Hence, the quantity we are interested in, is a value of v_n in the initial state:

$$V_n = v_n(q_0, 0).$$

By definition of runs of a timed automaton, we obtain the following language equations:

$$\begin{aligned} L_0(q, \mathbf{x}) &= \varepsilon; \\ L_{k+1}(q, \mathbf{x}) &= \bigcup_{(q,a,\mathfrak{g},\mathfrak{r},q') \in \Delta} \bigcup_{\tau : \mathbf{x}+\tau \in \mathfrak{g}} \tau a L_k(q', \mathfrak{r}(\mathbf{x}+\tau)). \end{aligned}$$

Since the automaton is deterministic, the union over transitions (the first $\bigcup$ in the formula) is disjoint. Hence, it is easy to pass to volumes:

$$v_0(q, \mathbf{x}) = 1; \tag{4.1}$$

$$v_{k+1}(q, \mathbf{x}) = \sum_{(q,a,\mathfrak{g},\mathfrak{r},q') \in \Delta} \int_{\tau : \mathbf{x}+\tau \in \mathfrak{g}} v_k(q', \mathfrak{r}(\mathbf{x}+\tau))\, d\tau. \tag{4.2}$$

Remark that for a fixed location q, and within every clock region, as defined in [3], the integral over $\tau : \mathbf{x} + \tau \in \mathfrak{g}$ can be decomposed into several $\int_l^u$ with bounds l and u either constants or of the form $c - x_i$ with c an integer and x_i a clock variable.

Also remark that v_0, the volume function for paths of length 0, has value 1 on every state, which may look like an arbitrary choice or a convention for defining an Euclidian 0-volume. However, paths of fleshy transitions going to an accepting state must yield a positive value, and this value has to be obtained by integrating v_0 several times. This implies that v_0 should have positive values in all accepting states, i.e. in all states. Moreover, if this positive value is 1, the integral formula is consistant with the conventional definition of the Euclidian n-volume for $n \geq 1$. But it is clear from the subsequent results that any other choice of positive values for v_0 would yield the same asymptotical growth rate, i.e. the same entropy.

These formulas lead to the following structural description of $v_n(q, \mathbf{x})$, which can be proved by a straightforward induction.

Lemma 1. *The function $v_n(q, \mathbf{x})$ restricted to a location q and a clock region can be expressed by a polynomial of degree n with rational coefficients in variables $\mathbf{x}$.*

Thus in order to compute the volume V_n one should find by symbolic integration polynomial functions $v_k(q, \mathbf{x})$ for $k = 0..n$, and finally compute $v_n(q_0, 0)$.

Theorem 1. *For a timed automaton $\mathcal{A}$ satisfying A1-A3, the volume V_n is a rational number, computable from $\mathcal{A}$ and n using the procedure described above.*

4.3 Operator Approach

In this central section of the chapter, we develop an approach to volumes and entropy of languages of nice timed automata based on functional analysis, first introduced in [7].

We start in 4.3.1 by identifying a functional space $\mathcal{F}$ containing the volume functions v_n. Next, we show that these volume functions can be seen as iterates of some positive integral operator Ψ on this space applied to the unit function (Sect. 4.3.2). We explore some elementary properties of this operator and its iterates in 4.3.3. This makes it possible to apply in 4.3.4 the theory of positive operators to Ψ and to deduce the main theorem of this chapter stating that the entropy equals the logarithm of the spectral radius of Ψ.

4.3.1 The Functional Space of a TA

In order to use the operator approach we first identify the appropriate functional space $\mathcal{F}$ containing volume functions v_n.

We define S as the disjoint union of all the entry regions of all the states of $\mathcal{A}$. Formally, $S = \{(q, \mathbf{x}) \mid \mathbf{x} \in \mathbf{r}_q\}$. The elements of the space $\mathcal{F}$ are bounded continuous functions from S to $\mathbb{R}$. The uniform norm $\|u\| = \sup_{\xi \in S} |u(\xi)|$ can be defined on $\mathcal{F}$,

yielding a Banach space structure. We can compare two functions in $\mathcal{F}$ pointwise, thus we write $u \leq v$ if $\forall \xi \in S : u(\xi) \leq v(\xi)$. For a function $f \in \mathcal{F}$ we sometimes denote $f(p, x)$ by $f_p(x)$. Thus, any function $f \in \mathcal{F}$ can be seen as a finite collection of functions f_p defined on entry regions $\mathbf{r}_p$ of locations of $\mathcal{A}$. The volume functions v_n (restricted to S) can be considered as elements of $\mathcal{F}$.

4.3.2 Volumes Revisited

Let us consider again the recurrent formula (4.2). It has the form $v_{k+1} = \Psi v_k$, where Ψ is the positive linear operator on $\mathcal{F}$ defined by the equation:

$$\Psi f(q, \mathbf{x}) = \sum_{(q,a,\mathfrak{g},\mathfrak{r},q') \in \Delta} \int_{\mathbf{x}+\tau \in \mathfrak{g}} f(q', \mathfrak{r}(\mathbf{x} + \tau))\, d\tau. \tag{4.3}$$

We have also $v_0 = 1$. Hence $v_n = \Psi^n 1$, and the problem of computing volumes and entropy is now phrased as studying iterations of a positive bounded linear operator Ψ on the functional space $\mathcal{F}$. The theory of positive operators guarantees, that under some hypotheses, v_n is close in direction to a positive eigenvector v^* of Ψ, corresponding to its leading eigenvalue ρ. Moreover, values of v_n will grow/decay exponentially like ρ^n. In the sequel, we refer to the book [38] when a result concerning positive operators is needed.

4.3.3 Exploring the Operator Ψ

Let us first state some elementary properties of this operator, starting by rewriting (4.3) as an operator on $\mathcal{F}$ and separating all its summands.

$$(\Psi f)_q(\mathbf{x}) = \sum_{\delta=(q,\ldots,q') \in \Delta} (\psi_\delta f_{q'})(\mathbf{x}). \tag{4.4}$$

For $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q')$ the operator ψ_δ acts from the space $C(\mathbf{r}_{q'})$ of bounded continuous functions on the target region to the space $C(\mathbf{r}_q)$ of functions on the source region. It is defined by the integral:

$$\psi_\delta f(\mathbf{x}) = \int_{\mathbf{x}+\tau \in \mathfrak{g}} f(\mathfrak{r}(\mathbf{x} + \tau))\, d\tau.$$

Iterating (4.4), we obtain a formula for powers of operator Ψ

$$(\Psi^k f)_p(\mathbf{x}) = \sum_{\delta_1 \ldots \delta_k \text{ from } p \text{ to } p'} (\psi_{\delta_1} \ldots \psi_{\delta_k} f_{p'})(\mathbf{x}). \tag{4.5}$$

Now we need some results on the iterations of ψ_δ. For this, first we state some useful properties of ψ_δ and its partial derivatives:

Proposition 6. *For any $f \in C(\mathbf{r}_q)$:*

1. *If $f \geq 0$ and f is not identically 0 then $\psi_\delta f$ is not identically 0.*

2. *$\|\psi_\delta f\| \leq \|f\|$ (in other words, $\|\psi_\delta\| \leq 1$).*

3. *If δ resets x_i then $\psi_\delta f$ is continuously differentiable by x_i and $\|\frac{\partial}{\partial x_i}\psi_\delta f\| \leq 2\|f\|$*

4. *If δ does not reset x_i and f is continuously differentiable by x_i, then $\psi_\delta f$ is continuously differentiable by x_i and $\|\frac{\partial}{\partial x_i}\psi_\delta f\| \leq 2\|f\| + \|\frac{\partial}{\partial x_i} f\|$.*

Proof.

(1) Let $\mathbf{x}_1 \in \mathbf{r}_{q'}$ be such that $f(\mathbf{x}_1) > 0$.

By B2 and B3, we know that there exists $\mathbf{x}_2 \in \mathfrak{g}$ such that $\mathfrak{r}(\mathbf{x}_2) = \mathbf{x}_1$. As $\mathbf{x}_2 \in \mathfrak{g}$, there also exists $x_3 \in \mathbf{r}_q$ and $\tau \in \mathbb{R}_{\geq 0}$ verifying $\mathbf{x}_2 = \mathbf{x}_3 + \tau_0$.

Furthermore, because δ is fleshy, there exists τ_1 and τ_2, $\tau_1 < \tau_2$, such that for every $\tau \in [\tau_1, \tau_2]$, $x_3 + \tau \in \mathfrak{g}$.

Put together, the integration interval of $\psi_\delta f(\mathbf{x}_3) = \int_{\mathbf{x}_3+\tau\in\mathfrak{g}} f(\mathfrak{r}(\mathbf{x}_3+\tau))\, d\tau$ contains a value τ_0, for which the integrated function is positive, and includes $[\tau_1, \tau_2]$, thus is neither empty nor a singleton. The integrated function being non-negative and continuous, its integral, $\psi_\delta f(\mathbf{x}_3)$, is positive.

(2) In $\psi_\delta f(\mathbf{x}) = \int_{\mathbf{x}+\tau\in\mathfrak{g}} f(\mathfrak{r}(\mathbf{x}+\tau))\, d\tau$, we estimate $|f(\cdot)|$ from above by the constant $\|f\|$ and the length of the integration interval by 1, as it is included in the region $\mathfrak{g}$. This gives us the requested bound.

(3) As $\mathfrak{r}$ resets x_i, $f(q', \mathfrak{r}(\mathbf{x} + \tau))$ does not depend on x_i, and thus $\psi_\delta(q, x) = \int_{\mathbf{x}+\tau\in\mathfrak{g}} f(q', \mathfrak{r}(\mathbf{x}+\tau))\, d\tau$ is differentiable by x_i. Its derivative is

$$\frac{\partial}{\partial x_i}\psi_\delta f(q, \mathbf{x}) = \frac{\partial}{\partial x_i}\int_{\mathbf{x}+\tau\in\mathfrak{g}} f(q', \mathfrak{r}(\mathbf{x}+\tau))\, d\tau \tag{4.6}$$

$$= \pm(f(q', \mathfrak{r}(\mathbf{x} + \tau_{max}) - f(q', \mathfrak{r}(\mathbf{x} + \tau_{min}))). \tag{4.7}$$

The choice of $+$ or $-$ sign in the line (4.7) and the bounds τ_{max} and τ_{min} depend on the form of the guard.

First, observe that the latter term is a sum of continuous functions and, as such, is continuous. Furthermore, this term is bounded in absolute value by $2\|f\|$. Thus, we prove $|\frac{\partial}{\partial x_i}\psi_\delta f(q, \mathbf{x})| \leq 2\|f\|$.

(4) As f is differentiable by x_i, then so is $\psi_\delta f(q, \mathbf{x}) = \int_{\mathbf{x}+\tau\in\mathfrak{g}} f(q', \mathfrak{r}(\mathbf{x}+\tau))\, d\tau$. Let us

differentiate it:

$$\begin{aligned}\frac{\partial}{\partial x_i}\psi_\delta f(q,\mathbf{x}) =& \frac{\partial}{\partial x_i}\int_{\mathbf{x}+\tau\in\mathfrak{g}} f(q',\mathfrak{r}(\mathbf{x}+\tau))\,d\tau\\ \frac{\partial}{\partial x_i}\psi_\delta f(q,\mathbf{x}) =& \pm\left(f(q',\mathfrak{r}(\mathbf{x}+\tau_{max}) - f(q',\mathfrak{r}(\mathbf{x}+\tau_{min}))\right)\\ &+\int_{\mathbf{x}+\tau\in\mathfrak{g}} \frac{\partial}{\partial x_i} f(q',\mathfrak{r}(\mathbf{x}+\tau))\,d\tau.\end{aligned}$$

The resulting expression is still continuous in x_i. Indeed the newly added term in the last equality is an integral of a continuous function that does not depend on x_i on an interval that continuously depends on x_i.

We already stated that $|(f(q',\mathfrak{r}(\mathbf{x}+\tau_{max}) - f(q',\mathfrak{r}(\mathbf{x}+\tau_{min})))|$ is smaller than $2\|f\|$. Also in $\int_{\mathbf{x}+\tau\in\mathfrak{g}} \frac{\partial}{\partial x_i} f(q',\mathfrak{r}(\mathbf{x}+\tau))\,d\tau$, we can estimate the integrated function from above by the norm $\|\frac{\partial}{\partial x_i}f\|$. As the integration interval is smaller than 1, the integral is smaller than this norm too. Hence, the required inequality holds: $|\frac{\partial}{\partial x_i}\psi_\delta f(q,\mathbf{x})| \leq 2\|f\| + \|\frac{\partial}{\partial x_i}f\|$.

□

Now, we can prove the following result on the powers of Ψ.

Proposition 7. *Consider operator Ψ.*

1. *If $f \geq 0$ is not zero on p' and there is a path of length k from p to p' then $\Psi^k f$ is not identically zero on p.*

2. *For D defined in assumption A4 there exists a constant $E \in \mathbb{R}$ such that for any $f \in \mathcal{F}$ the following estimate hold:*

$$\forall i : \left\|\frac{\partial}{\partial x_i}\Psi^D f\right\| \leq E\|f\|.$$

Proof.

(1) This is a straightforward induction using (4.5) and Prop. 6-1.

(2) For some x_i, and a location p, the following equality holds:

$$\frac{\partial}{\partial x_i}(\Psi^D f)_p(\mathbf{x}) = \sum_{\delta_1\ldots\delta_D \text{ from } p \text{ to } p'} \frac{\partial}{\partial x_i}(\psi_{\delta_1}\ldots\psi_{\delta_D} f_{p'})(\mathbf{x}).$$

Let us consider one term of this sum corresponding to one path. By hypothesis, in this path, there is a first transition δ_k, $1 \leq k \leq D$, such that δ_k resets x_i.

By Prop. 6-3, $\psi_{\delta_k}\ldots\psi_{\delta_D} f_{p'}$ is continuously differentiable by x_i. By induction and using Prop. 6-4, it follows that $\psi_{\delta_1}\ldots\psi_{\delta_k}\ldots\psi_{\delta_D} f_{p'}$ is also continuously differentiable by x_i.

Now we differentiate this term. For every j, $1 \leq j \leq D$, iterating Prop. 6-2 $D - j$ times, we obtain $\left\|\psi_{\delta_j} \dots \psi_{\delta_D} f_{p'}\right\| \leq \|f\|$. Thus, by Prop. 6-3, we have $\left\|\frac{\partial}{\partial x_i}\psi_{\delta_k} \dots \psi_{\delta_D} f_{p'}\right\| \leq 2\|f\|$. It follows by induction on the path, using Prop. 6-4, that $\left\|\frac{\partial}{\partial x_i}\psi_{\delta_1} \dots \psi_{\delta_k} \dots \psi_{\delta_D} f_{p'}\right\| \leq 2k\|f\|$.

Now, if we come back to the sum, we have, at least, the following bound:

$$\left\|\frac{\partial}{\partial x_i}(\Psi^D f)_p\right\| \leq 2d^D D\|f\|,$$

with d: maximal degree of the underlying graph of Δ, which is true for every p, therefore $\|\frac{\partial}{\partial x_i}\Psi^D f\| \leq 2d^D D\|f\|$.

□

Now we are ready to prove the following important property of Ψ:

Theorem 2. *The operator Ψ^D is compact on $\mathcal{F}$.*

Proof. Consider $\mathcal{B}$ – the unit ball of $\mathcal{F}$. Let us prove that $\Psi^D\mathcal{B}$ is a compact set. This set is clearly bounded. It follows from Prop. 7-2, that the whole set $\Psi^D\mathcal{B}$ is Lipschitz continuous with constant $E\#C$, where $\#C$ is the dimension of the clock space. Hence it is equicontinuous, and, by Arzela-Ascoli theorem, compact. □

Next two lemmata will be used in the proof of the Main Theorem. Denote by ρ the spectral radius of Ψ.

Lemma 2. *If $\rho > 0$ then it is an eigenvalue of Ψ with an eigenvector $v^* \geq 0$.*

of Lemma. According to Thm. 9.4 of [38] the statement holds for every positive linear operator with a compact power. Thus, the result follows immediately from Thm. 2. □

Lemma 3. *If $\rho > 0$ then the eigenvector v^* satisfies $v^*(q_0, 0) > 0$.*

Proof. Let $(p, \mathbf{x})$ be a state for which v^* is positive. Consider a path from $(q_0, 0)$ to $(p, \mathbf{x})$, and let k be its length. By Prop. 7-1, the function $\Psi^k v^*$ is not identically zero on the region of $(q_0, 0)$. Since this region is a singleton, this means that $(\Psi^k v^*)(q_0, 0) > 0$. Since v^* is an eigenvector, we rewrite this as $\rho^k v^*(q_0, 0) > 0$, and the statement is immediate.

□

4.3.4 Main Theorem

The main result of this chapter can now be stated.

Theorem 3. *For any nice TA $\mathcal{A}$ the entropy $\mathcal{H}$ of its language coincides with logarithm of the spectral radius of the Ψ operator defined on $\mathcal{F}$.*

Proof. Notice that

$$V_n = v_n(q_0; 0) \leq \|v_n\| = \|\Psi^n 1\| \leq \|\Psi^n\|.$$

Taking logarithm and dividing by n, we obtain $\log V_n/n \leq \log \|\Psi^n\|/n$.

The limit of the right-hand side is $\log \rho$ due to Gelfand's formula for spectral radius: $\rho = \lim_{n\to\infty} \|\Psi^n\|^{1/n}$. Thus, we obtain the required upper bound for the entropy:

$$\mathcal{H} = \limsup_{n\to\infty} \log V_n/n \leq \log \rho.$$

In the case when $\rho > 0$ we also have to prove the lower bound. In this case Lemma 2 applies and an eigenvector $v^* \geq 0$ with norm 1 exists. This yields the inequality $v^* \leq 1$, to which, for any natural n, we can apply the positive operator Ψ^n. Using the fact that v^* is an eigenvector and the formula for v_n we obtain $\rho^n v^* \leq v_n$. Then, taking the values of the functions in the initial state we get $\rho^n v^*(q_0; 0) \leq V_n$. Hence, by Lemma 3, denoting the positive number $v^*(q_0; 0)$ by δ: $\rho^n \delta \leq V_n$. Taking logarithm, dividing by n, and taking the limit we obtain:

$$\log \rho \leq \liminf_{n\to\infty} \log V_n/n = \mathcal{H}.$$

□

The following result is immediate from the proof of the Theorem.

Corollary 1. *For any nice TA $\mathcal{A}$ the* lim sup *in the definition of the entropy is in fact a limit, that is $\mathcal{H} = \lim_{n\to\infty} \log V_n/n$.*

After stating those important results, some remarks come to mind.

Remark 2. *The proof of Thm. 3 shows how important it was that every state was accepting: it was possible to find an eigenvector smaller than V_0 for the leading eigen value, because all accepting entry regions had positive 0-volumes. Working on languages that are not prefix closed would require a finer analysis of iterates of Ψ.*

Remark 3. *The proof of the entropy lower bound makes use of Lem. 2 and Lem. 3 for the case $\rho > 0$. A too quick analysis would tend to suggest that we are always in this case when A4 is met. However at least one exemple can be shown (Fig. 4.5), of a nice automaton yielding a spectral radius of value 0. This example shows that phenomena comparable to the Zenoness in Fig. 4.3 can still happen when A4 is met. However A4 is enough for proving the theorem.*

4.4 Computing the Entropy

The characterization of $\mathcal{H}$ in Theorem 3 solves the main problem explored in this chapter, but its concrete application requires computing the spectral radius of an integral operator Ψ, and this is not straightforward. In 4.4.1, we solve this problem for a subclass of automata by reduction to differential equations. As for the general situation,

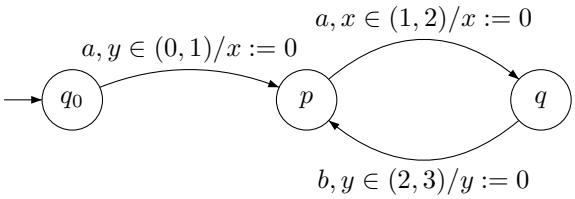


Figure 4.5: The automaton on this figure is nice and its region-split form is even fleshy. However, as time progresses to infinity, the time difference between two a's or two b's has to become closer and closer to 2 because every step reduces the possible time span for the next event to occur. Using the symbolic technics described in the next section we find that the spectral radius of the operator Ψ actually is 0.

in 4.4.2 we give an iterative procedure, which approximates the spectral radius and the entropy with a guaranteed precision.

4.4.1 Case of "$1\frac{1}{2}$ Clock" Automata

Consider now the subclass of (fleshy region-split) automata with entry regions of all the locations having dimension 0 or 1. In other words, in such automata for every discrete transition there is at most one clock non reset. We call this class the class of $1\frac{1}{2}$ *clock automata*. The idea of the symbolic algorithm for computing the entropy of such automata is presented in Table 3.

Algorithm 3 for computing $\mathcal{H}$ symbolically on $1\frac{1}{2}$ clocks automata (idea)

---

1. Transform $\mathcal{A}$ into the fleshy region-split form and check that it has $1\frac{1}{2}$ clock.
2. Write the integral eigenvalue equation (I) with one variable.
3. Derivate (I) w.r.t. x and get a differential equation (D).
4. Instantiate (I) at 0, and obtain a boundary condition (B).
5. Solve (D) with boundary condition (B).
6. Take $\rho = \max\{\lambda |$ a non-0 solution exists$\}$.
7. Return $\mathcal{H}(L(\mathcal{A})) = \log \rho$.

Notice first that the set $S = \{(q, \mathbf{x}) \mid \mathbf{x} \in \mathbf{r}_q\}$ is now a disjoint union of unit length intervals and singleton points. After a change of variables, each of those unit intervals can be represented as $x \in (0; 1)$, and a singleton point as $x = 0$. In both cases x is a scalar variable, equal in the first case to the fractional part of x_q, where $x_q \in C$ is

the only clock whose value is positive in $\mathbf{r}_q$. Thus, every $f \in F$ can be seen as a finite collection of functions f_q of one scalar argument: x.

In this case the expression of the operator ψ_δ, corresponding to one transition $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q')$, can be made more explicit. First we recall the definition of ψ_δ:

$$\psi_\delta f(\mathbf{x}) = \int_{\mathbf{x}+\tau \in \mathfrak{g}} f(\mathfrak{r}(\mathbf{x}+\tau))\, d\tau.$$

A careful but straightforward analysis shows that from the entry region of every state q, non-degenerated regions of two types are alternatively visited: regions where x_q is greater than the other clocks, and regions where it is not.

Assuming t is the difference between the fractional parts of x'_q and x_q, for guards $\mathfrak{g}$ that are regions of the first type (a), $x + \tau \in \mathfrak{g}$ is equivalent to $t \in (0, 1-x)$, and for the other type (b), it is equivalent to $t \in (1-x, 1)$.

Furthermore, the reset function $\mathfrak{r}$ can behave in three different ways: either it resets every clock but one that is not x_q (1), or it resets every clock but x_q (2), or it resets every clock (3).

Those two criteria can be combined in 6 different ways, partitioning the set of transitions starting from q in as many sets: $\Delta_{qa1}, \Delta_{qb1}, \Delta_{qa2}, \Delta_{qb2}, \Delta_{qa3}$ and Δ_{qb3}, such that Ψ can now be written the following way:

$$\begin{aligned}
\Psi f(q,x) = &\sum_{\delta\in\Delta_{qa1}} \int_0^{1-x} f(q', x+t)dt && + \sum_{\delta\in\Delta_{qb1}} \int_{-x}^{0} f(q', x+t)dt \\
&+ \sum_{\delta\in\Delta_{qa2}} \int_0^{1-x} f(q', t)dt && + \sum_{\delta\in\Delta_{qb2}} \int_{1-x}^{1} f(q', t)dt \\
&+ \sum_{\delta\in\Delta_{qa3}} (1-x) f(q', 0) && + \sum_{\delta\in\Delta_{qb3}} x f(q', 0).
\end{aligned}$$

Now we define the square matrices D_{ij} such that the operator can be written as follows:

$$\begin{aligned}
\Psi f(x) = \; & D_{a1} \textstyle\int_0^{1-x} f(x+t)dt && + && D_{b1} \textstyle\int_{-x}^{0} f(x+t)dt \\
& + D_{a2} \textstyle\int_0^{1-x} f(t)dt && + && D_{b2} \textstyle\int_{1-x}^{1} f(t)dt \\
& + D_{a3}(1-x) f(0) && + && D_{b3} x f(0).
\end{aligned}$$

This is the explicit formula for Ψ we have been looking for. Now, computing the entropy of the language of the automaton using Thm. 3 involves finding the leading eigenvalue of Ψ, that is the greatest $\lambda \in \mathbb{R}$ such that for some non-zero function $f \in \mathcal{F}$:

$$\Psi f = \lambda f. \tag{4.8}$$

We will solve this by transforming this equality into a differential equation. A smooth function $h : [0,1] \to \mathbb{R}$ equals 0 iff $h(0) = 0$ and $h'(x) = 0$ for all $x \in (0,1)$.

Applying this to $(\Psi f - \lambda f)$[6] we obtain that (4.8) is equivalent to the differential equation

$$\lambda f'(x) = \quad (D_{b1} - D_{a1})f(x) + (D_{b2} - D_{a2})f(1-x) + (D_{b3} - D_{a3})f(0). \tag{4.9}$$

with boundary condition

$$\lambda f(0) = (D_{a1} + D_{a2}) \int_0^1 f(t)dt + D_{a3} f(0). \tag{4.10}$$

Now we solve the differential equation (4.9) by introducing the functions u and w as follows: $u(x) = f(x) + f(1-x)$ and $w(x) = f(x) - f(1-x)$. This removes the cumbersome dependency between f and $x \mapsto f(1-x)$ and enables us to rewrite the previous equation as a differential system:

$$\begin{cases} \lambda u'(x) &= Aw(x) \\ \lambda w'(x) &= Bu(x) + C(u(0) + w(0)) \\ w(\frac{1}{2}) &= 0 \end{cases}, \tag{4.11}$$

where $A \stackrel{def}{=} D_{b1} - D_{a1} - D_{b2} + D_{a2}$, $B \stackrel{def}{=} D_{b1} - D_{a1} + D_{b2} - D_{a2}$ and $C \stackrel{def}{=} D_{b3} - D_{a3}$.

Note that due to the properties of the functions u and w, this system has to be considered on the interval $[0, \frac{1}{2}]$ only, and $w(\frac{1}{2}) = 0$ is the consequence of the definition of w. This rewriting is without loss of information, as the original equation (4.9) can be recovered by adding those two equations term by term.

System (4.11) implies

$$\begin{cases} \lambda^2 w''(x) &= BAw(x) \\ \lambda u'(x) &= Aw(x) \\ w(\frac{1}{2}) &= 0. \end{cases} \tag{4.12}$$

The first equation of (4.12) is homogeneous and has a solution space of dimension $2n$, but using the fact that $w(\frac{1}{2}) = 0$, this allows us to consider only n independent solutions w_i.

Using the second equation, we get $u(x) = \frac{1}{\lambda}\int_0^x Aw(t)dt + u_0$, for every solution w of the first equation and every $u_0 \in \mathbb{R}^n$. Thus (4.12) yields a solution space of dimension $2n$.

Now having a solution (u, w) to (4.12) implies that $\lambda^2 w''(x) = BAw(x)$, which implies $\lambda w'(x) = \frac{1}{\lambda}\int_0^x BAw(t)dt + \lambda w'(0)$ and thus

$$\lambda w'(x) = \int_0^x Bu'(t)dt + \lambda w'(0) = Bu(x) - Bu(0) + \lambda w'(0).$$

Therefore (u, w) is also a solution to (4.11) if and only if $C(u(0) + w(0)) + Bu(0) = \lambda w'(0)$. Coming back to (4.9), $f \triangleq \frac{u+w}{2}$ is a solution to this system if and only if

$$\lambda(f(0) - f(1)) = 2Cf(0) + B(f(0) + f(1)),$$

[6] It is easy to see that for eigenfunctions f this expression should be smooth and well-defined in 0 and 1.

or again

$$(\lambda - B - 2C)f(0) = (\lambda + B)f(1). \tag{4.13}$$

To sum up, the dimension of the space of the solutions of (4.12) is $2n$, thus so is the space S of the functions $f = \frac{u+w}{2}$ such that (u, w) is solution to (4.12). This allows us to write every such f as FM where F is an $n \times 2n$ matrix whose columns are a basis of S, and M is a vertical vector of $\mathbb{R}^{2n}$.

Every such $f = FM$ is a solution of (4.8) if and only if both (4.10) and (4.13) hold, which we rewrite here, replacing f by FM:

$$\begin{cases} \lambda F(0)M = ((D_{a1} + D_{a2})(\int_0^1 F(t)dt) + D_{a3}F(0))M \\ (\lambda - B - 2C)F(0)M = (\lambda + B)F(1)M \end{cases}$$

$$\begin{cases} (F(0) - ((D_{a1} + D_{a2})(\int_0^1 F(t)dt) + D_{a3}F(0)))M = 0 \\ ((\lambda - B - 2C)F(0) - (\lambda + B)F(1))M = 0. \end{cases}$$

Considering this as an equation on M, this homogeneous linear system has non-zero solutions if and only if

$$\det \begin{pmatrix} F(0) - ((D_{a1} + D_{a2})(\int_0^1 F(t)dt) + \Delta_{a3}F(0)) \\ (\lambda - B - 2C)F(0) - (\lambda + B)F(1) \end{pmatrix} = 0$$

This is a transcendental equation on λ (as $F(x)$ has coefficients which are polynomials of complex exponentials of $\frac{x}{\lambda}$) that can be solved numerically, and which we know to have a maximal real solution, which is also the spectral radius of Ψ (Lem. 2). The logarithm of this value is the entropy of the language (Thm. 3).

Summing up all those computations yields the complete algorithm for automata with $1\frac{1}{2}$ clocks depicted in Table 4.

Algorithm 4 for computing $\mathcal{H}$ symbolically on $1\frac{1}{2}$ clocks automata (concrete version)

---

1. Transform $\mathcal{A}$ into the fleshy region-split form and check that it has $1\frac{1}{2}$ clock.
2. Compute the matrices D_{ij} and next A, B, C.
3. Deduce the general solution FM to (4.9).
4. Find the greatest root ρ (w.r.t. the unknown λ) of

$$\det \begin{pmatrix} F(0) - ((D_{a1} + D_{a2})(\int_0^1 F(t)dt) + \Delta_{a3}F(0)) \\ (\lambda - B - 2C)F(0) - (\lambda + B)F(1). \end{pmatrix}$$

5. Then we have $\mathcal{H}(L(\mathcal{A})) = \log \rho$.

Application to the Running Example

We apply the method just described to compute the entropy of the language of the automaton $\mathcal{A}_3$ of Fig. 4.1 which is a "$1\frac{1}{2}$ clocks" one. Its fleshy region-split form is presented on Fig. 4.4.

By symmetry, the volume of a path of length $n \in \mathbb{N}$ is the same function v_n in both non-initial states. Thus v_n is characterized by:

$$\begin{cases} v_0(x) = 1 \\ v_{n+1}(x) = (\Psi v_n)(x) \triangleq \int_0^{1-x} v_n(t)dt. \end{cases}$$

According to Thm. 3, the entropy can be found as $\log \rho(\Psi)$, and by Lemma 2 $\rho(\Psi)$ is the maximal eigenvalue of Ψ. Let us write the eigenvalue equation:

$$\lambda v(x) = \int_0^{1-x} v(t)dt. \tag{4.14}$$

Differentiating it twice w.r.t x we get:

$$\lambda v'(x) = -v(1-x) \tag{4.15}$$
$$\lambda^2 v''(x) = -v(x) \tag{4.16}$$

The solutions have the form $v(x) = \alpha \sin(\frac{x}{\lambda}) + \beta \cos(\frac{x}{\lambda})$. Using (4.14) with $x = 1$ we find $v(1) = 0$. We inject this in (4.15) for $x = 0$ and deduce $\alpha = 0$. Thus $v(x) = \beta \cos(\frac{x}{\lambda})$ and $\cos(\frac{1}{\lambda}) = 0$. This implies that the solutions correspond to $\lambda = \frac{2}{(2k+1)\pi}$ with $k \in \mathbb{Z}$. The highest of those is $\lambda = 2/\pi$, and we can verify that $v(x) = \cos(\frac{x\pi}{2})$ satisfies $\frac{2}{\pi} v = \Psi v$. Therefore $\rho(\Psi) = 2/\pi$, and the entropy of this automaton is $\log(2/\pi)$.

Notice this value is negative, this very fact would be unsettling with respect to usual definitions of entropy. Here, the reader should recall that this entropy is only the exponent of the asymptotical growth of the n-volume of the language of the automaton. However, the relation with Kolmogorov complexity in the sequel will give a relevant information theoretical interpretation of the value of this entropy.

4.4.2 General Case

If several clocks are not reset in some transitions, then the entry regions are multi-dimensional, and the volume functions therefore depend on several real variables. Hence, we cannot reduce the integral equation to an ordinary differential equation and it is not even known if all integral symbols can be eliminated using partial derivation. This makes it difficult to find the eigenfunction symbolically. Instead, we can use standard iterative procedures for eigenvalue approximation for positive operators. Recall that the volume function satisfies $v_n = \Psi^n 1$. The following theorem is close to Thms. 16.1-16.2 from [38].

Theorem 4. *If for some $\alpha, \beta \in \mathbb{R}, m \in \mathbb{N}$ the following inequality holds: $\alpha v_m \leq v_{m+1} \leq \beta v_m$, and the volume $V_m = v_m(q_0, 0) > 0$, then* $\log \alpha \leq \mathcal{H} \leq \log \beta$.

Proof. Applying the positive operator Ψ^n to the inequalities $\alpha v_m \leq v_{m+1} \leq \beta v_m$, and using the formula $v_n = \Psi^n 1$ we obtain that for all n

$$\alpha v_{m+n} \leq v_{m+n+1} \leq \beta v_{m+n}.$$

From this by induction, we prove that for all n

$$\alpha^n v_m \leq v_{m+n} \leq \beta^n v_m.$$

We apply this to the initial state $(q_0, 0)$ (remember that $V_n = v_n(q_0, 0)$):

$$\alpha^n V_m \leq V_{m+n} \leq \beta^n V_m.$$

Take a logarithm, divide by $m + n$ and take a $\limsup_{n\to\infty}$ (remember that $\mathcal{H} = \limsup_{n\to\infty} \log V_n/n$):

$$\log \alpha \leq \mathcal{H} \leq \log \beta$$

(we have used the fact that $V_m > 0$).

□

This theorem yields a procedure[7] to estimate $\mathcal{H}$ summarized in Table 5.

Algorithm 5 bounding $\mathcal{H}$ by iterating Ψ

1. Transform $\mathcal{A}$ into the fleshy region-split form.
2. Choose an m and compute symbolically the piecewise polynomial functions v_m and v_{m+1}.
3. Check that $v_m(q_0, 0) > 0$.
4. Compute $\alpha = \min(v_{m+1}/v_m)$ and $\beta = \max(v_{m+1}/v_m)$.
5. Conclude that $\mathcal{H} \in [\log \alpha; \log \beta]$.

Example: Again $\mathcal{A}_3$

We apply the iterative procedure above to our running example $\mathcal{A}_3$. As explained in Sect. 4.4.1, we can just consider the operator on $C(0; 1)$

$$\Psi f(x) = \int_0^{1-x} f(s)\, ds.$$

The iteration results are given in Table 4.1.

Experiments show that the bounds usually converge quite fast when A5 holds and the automaton is strongly connected. However, a proof of convergence and of its speed remains to be found.

[7] One possible optimization is to compute α and β separately on every strongly connected reachable component of the automaton, and take the maximal values.

m	$v_m(x)$	α	β	$\log\alpha$	$\log\beta$
0	1	0	1		
1	$1-x$	0.5	1	-1	0
2	$1-x-1/2\,(1-x)^2$	0.5	0.667	-1	-0.584
3	$1/2\,(1-x)-1/6\,(1-x)^3$	0.625	0.667	-0.679	-0.584
4	$1/3\,(1-x)+1/24\,(1-x)^4-1/6\,(1-x)^3$	0.625	0.641	-0.679	-0.643
5	$\frac{5}{24}\,(1-x)+\frac{1}{120}\,(1-x)^5-1/12\,(1-x)^3$	0.6354	0.641	-0.6543	-0.643
6	$\frac{2}{15}\,(1-x)-\frac{1}{720}\,(1-x)^6+\frac{1}{120}\,(1-x)^5-\frac{1}{18}\,(1-x)^3$	0.6354	0.6371	-0.6543	-0.6506
7	$\frac{61}{720}\,(1-x)-\frac{1}{5040}\,(1-x)^7+\frac{1}{240}\,(1-x)^5-\frac{5}{144}\,(1-x)^3$	0.6364	0.6371	-0.6518	-0.6506

Table 4.1: Iterating the operator for $\mathcal{A}_3$ ($\mathcal{H}=\log(2/\pi)\approx\log 0.6366\approx -0.6515$)

4.5 Discretization Approach

4.5.1 Discretizing the Volumes

Another approach we published in [8], is to do entropy computation is by discretization. This approach also sheds a new light on the information-theoretic interpretation of entropy. The discretizations of timed languages we use are strongly inspired by [10, 33].

4.5.2 ε-words and ε-balls

We start with a couple of preliminary definitions. Take an $\varepsilon=1/N>0$. A timed word w is ε-timed if all the delays in this word are multiples of ε. Any ε-timed word w over an alphabet Σ can be written as $w=h_\varepsilon(v)$ for an untimed $v\in\Sigma\cup\{\tau\}$, where the morphism h_ε is defined as follows:

$$h_\varepsilon(a)=a \text{ for } a\in\Sigma, \qquad h_\varepsilon(\tau)=\varepsilon.$$

The discrete word v with ticks τ (standing for ε delays) represents in this way the ε-timed word w.

Example Let $\varepsilon=1/5$, then the timed word $0.6a0.4ba0.2a$ is ε-timed. Its representation is $\tau\tau\tau a\tau\tau ba\tau a$.

The notions of ε-timed words and their representation can be ported straightforwardly to languages.

For a timed word $w=t_1a_1t_2a_2\ldots t_na_n$ we introduce its North-East ε-neighbourhood like this:

$$\mathcal{B}_\varepsilon^{NE}(w)=\{s_1a_1s_2a_2\ldots s_na_n \mid \forall i\,(s_i\in[t_i;t_i+\varepsilon])\}\,.$$

For a language L, we define its NE-neighbourhood elementwise:

$$\mathcal{B}_\varepsilon^{NE}(L)=\bigcup_{w\in L}\mathcal{B}_\varepsilon^{NE}(w). \tag{4.17}$$

The next simple lemma will play a key role in our algorithm (here $\#L$ stands for the cardinality of L).

Lemma 4. *Let L be some finite set of timed words of length n. Then*

$$\mathrm{Vol}(\mathcal{B}_\varepsilon^{NE}(L)) \leq \varepsilon^n \# L.$$

If, moreover, L is ε-timed, then

$$\mathrm{Vol}(\mathcal{B}_\varepsilon^{NE}(L)) = \varepsilon^n \# L.$$

Proof. Notice that for a timed word w of a length n the set $\mathcal{B}_\varepsilon^{NE}(w)$ is a hypercube of edge ε (in the delay space), and of volume ε^n. Notice also that neighbourhoods of different ε-timed words are almost disjoint: the interior of their intersections are empty. With these two remarks, the two statements are immediate from (4.17). □

4.5.3 Discretizing Timed Languages and Automata

Suppose now that we have a timed language L recognized by a timed automaton $\mathcal{A}$ satisfying A2-A5 and we want to compute its entropy (or just the volumes V_n). Take an $\varepsilon = 1/N > 0$. We will build two ε-timed languages L_- and L_+ that under- and over-approximate L in the following sense:

$$\mathcal{B}_\varepsilon^{NE}(L_-) \subset L \subset \mathcal{B}_\varepsilon^{NE}(L_+). \tag{4.18}$$

The recipe is like this. Take the timed automaton $\mathcal{A}$ accepting L. Discrete automata A_+^ε and A_-^ε can be constructed in two stages. First, we build counter automata C_+^ε and C_-^ε. They have the same states as $\mathcal{A}$, but instead of every clock x they have a counter c_x (roughly representing x/ε). For every state add a self-loop labelled by τ and incrementing all the counters. Replace any reset of x by a reset of c_x. Whenever $\mathcal{A}$ has a guard $x \in [l; u]$ (or $x \in (l; u)$, or some other interval), the counter automaton C_+^ε has a guard $c_x \in [l/\varepsilon \dot{-} D; u/\varepsilon - 1]$ (always the closed interval) instead, where D is as in assumption A4. At the same time, C_-^ε has a guard $c_x \in [l/\varepsilon; u/\varepsilon - D]$. Automata C_+^ε and C_-^ε with bounded counters can be easily transformed into finite-state ones A_+^ε and A_-^ε .

Lemma 5. *Languages $L_+ = h_\varepsilon(L(A_+^\varepsilon))$ and $L_- = h_\varepsilon(L(A_-^\varepsilon))$ have the required property (4.18).*

Proof sketch.

Inclusion $\mathcal{B}_\varepsilon^{NE}(L_-) \subset L$. Let a discrete word $u \in L(A_-^\varepsilon)$, let $v = h_\varepsilon(u)$ be its ε-timed version, and let $w \in \mathcal{B}_\varepsilon^{NE}(v)$. We have to prove that $w \in L$. Notice first that $L(A_-^\varepsilon) = L(C_-^\varepsilon)$ and hence u is accepted by C_-^ε. Mimic the run of C_-^ε on u, but replace every τ by an ε duration, thus, a run of $\mathcal{A}$ on v can be obtained. Moreover, in this run every guard $x \in [l, u]$ is respected with a security margin: in fact, a stronger guard $x \in [l, u - D\varepsilon]$ is respected. Now one can mimic the same run of $\mathcal{A}$ on w. By definition of the neighbourhood, for any delay t_i in u the corresponding delay t'_i in w belongs to $[t_i, t_i + \varepsilon]$. Clock values are always sums of several (up

to D) consecutive delays. Whenever a narrow guard $x \in [l, u - D\varepsilon]$ is respected on v, its "normal" version $x' \in [l, u]$ is respected on w. Hence, the run of $\mathcal{A}$ on w obtained in this way respects all the guards, and thus $\mathcal{A}$ accepts w. We deduce that $w \in L$.

Inclusion $L \subset \mathcal{B}_\varepsilon^{NE}(L_+)$. First, we define an approximation function on $\mathbb{R}_+$ as follows:

$$\underline{t} = \begin{cases} 0 & \text{if} \quad t = 0 \\ t - \varepsilon & \text{if} \quad t/\varepsilon \in \mathbb{N}_+ \\ \varepsilon \lfloor t/\varepsilon \rfloor & \text{otherwise.} \end{cases}$$

Clearly, $\underline{t}$ is always a multiple of ε and belongs to $[t-\varepsilon, t)$ with the only exception that $\underline{0} = 0$.

Now we can proceed with the proof. Let $w = t_1 a_1 \dots t_n a_n \in L$. We define its ε-timed approximation v by approximating all the delays: $v = \underline{t_1} a_1 \dots \underline{t_n} a_n$. By construction $w \in \mathcal{B}_\varepsilon^{NE}(v)$. The run of $\mathcal{A}$ on w respects all the guards $x \in [l; u]$. Notice that the clock value of x on this run is a sum of several (up to D) consecutive t_i. If we try to run $\mathcal{A}$ over the approximating word v, the value x' of the same clock at the same transition would be a multiple of ε and it would belong to $[x - D\varepsilon; x)$. Hence $x' \in [l \dot{-} D\varepsilon, u - \varepsilon]$. By definition of C_+ this means that the word $u = h_\varepsilon^{-1}(v)$ is accepted by this counter automaton. Hence $v \in L_+$.

Let us summarize: for any $w \in L$, we have constructed $v \in L_+$ such that $w \in \mathcal{B}_\varepsilon^{NE}(v)$. This concludes the proof.

□

4.5.4 Counting Discrete Words

Once the automata A_+^ε and A_-^ε constructed, we can count the number of words with n events and its asymptotic behaviour using the following simple result.

Lemma 6. *Given an automaton $\mathcal{B}$ over an alphabet $\{\tau\} \cup \Sigma$, let*

$$L_n = L(\mathcal{B}) \cap (\tau^* \Sigma)^n .$$

Then (1) $\#L_n$ is computable; and (2) $\lim_{n\to\infty}(\log \#L_n/n) = \log \rho_{\mathcal{B}}$ with $\rho_{\mathcal{B}}$ a computable algebraic real number.

Proof. We proceed in three stages. First, we determinize $\mathcal{B}$ and remove all the useless states (unreachable from the initial state). These transformations yield an automaton $\mathcal{D}$ accepting the same language, and hence having the same cardinalities $\#L_n$. Since the automaton is deterministic, to every word in L_n corresponds a unique accepting path with n events from Σ and terminating with such an event.

Next, we eliminate the tick transitions τ. As we are counting paths, we obtain an automaton without silent (τ) transitions, but with multiplicities representing the number of realizations of every transition. More precisely, the procedure is as follows. Let

$\mathcal{D} = (Q, \{\tau\} \cup \Sigma, \delta, q_0)$. We build an automaton with multiplicities $\mathcal{E} = (Q, \{e\}, \Delta, q_0)$ over one-letter alphabet. For every $p, q \in Q$ the multiplicity of the transition $p \to q$ in $\mathcal{E}$ equals the number of paths from p to q in $\mathcal{D}$ over words from $\tau^*\Sigma$. A straightforward induction over n shows that the number of paths in $\mathcal{D}$ with n non-tick events equals the number of n-step paths in $\mathcal{E}$ (with multiplicities).

Let M be the adjacency matrix with multiplicities of $\mathcal{E}$. It is well known (and easy to see) that the $\#L(n)$ (that is the number of n-paths) can be found as the sum of the first line of the matrix M_-^n. This allows computing $\#L(n)$. Moreover, using Perron-Frobenius theorem we obtain that $\#L(n) \sim \rho^n$ where ρ is the spectral radius of M, the greatest (in absolute value) real root λ of the integer characteristic polynomial $\det(M - \lambda I)$. □

4.5.5 From Discretizations to Volumes

As soon as we know how to compute the cardinalities of under- and over- approximating languages $\#L_-(n)$ and $\#L_+(n)$ and their growth rates ρ_- and ρ_+, we can deduce the following estimates solving our problems.

Theorem 5. *For a timed automaton $\mathcal{A}$ satisfying A2-A5, the n-volumes of its language satisfy the estimates:*

$$\#L_-(n) \cdot \varepsilon^n \le V_n \le \#L_+(n) \cdot \varepsilon^n.$$

Proof. In inclusions (4.18) take the volumes of the three terms, and use Lemma 4. □

Theorem 6. *For a timed automaton $\mathcal{A}$ satisfying A2-A5, the entropy of its language satisfies the estimates:*

$$\log(\varepsilon\rho_-) \le \mathcal{H}(L(\mathcal{A})) \le \log(\varepsilon\rho_+).$$

Proof. Just use the previous result, take the logarithm, divide by n and pass to the limit. □

We summarize the algorithm in Table 6.

This theorem can be used to estimate the entropy. However, it can also be read in a converse direction: the cardinality of L restricted to n events and discretized with quantum ε is close to $2^{\mathcal{H}n}/\varepsilon^n$. Hence, we can encode $\mathcal{H} - \log\varepsilon$ bits of information per event. These information-theoretic considerations are made more explicit in Sect. 4.6 below.

A Case Study.

Consider the example $L_3 = \{t_1 a t_2 b t_3 a t_4 b \cdots \mid t_i + t_{i+1} \in [0; 1]\}$ from Subsect. 4.2.2. We need two clocks to recognize this language, and they are never reset together. We choose $\varepsilon = 0.05$ and build the automata on Fig. 4.6 according to the recipe (the discrete ones A_+ and A_- are too big to fit on the figure).

Algorithm 6 bounding $\mathcal{H}$ by discretization

1. Choose an $\varepsilon = 1/N$.
2. Build the counter automata C_-^ε and C_+^ε.
3. Transform them into finite automata A_-^ε and A_+^ε.
4. Eliminate τ transitions introducing multiplicities.
5. Obtain adjacency matrices M_- and M_+.
6. Compute their spectral radii ρ_- and ρ_+.
7. Conclude that $\mathcal{H} \in [\log \varepsilon\rho_-; \log \varepsilon\rho_+]$.

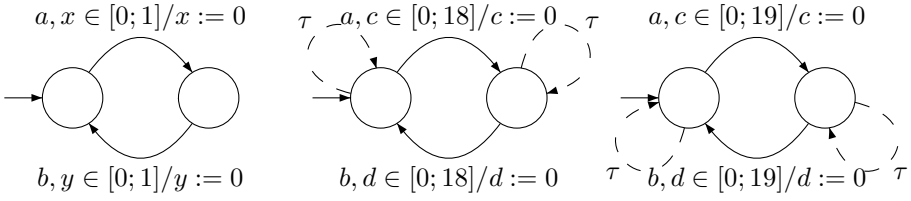


Figure 4.6: A two-clock timed automaton $\mathcal{A}_3$ and its approximations $C_-^{0.05}$ and $C_+^{0.05}$. All τ-transitions increment counters c and d.

We transform $C_-^{0.05}$ and $C_+^{0.05}$, into A_+ and A_-, eliminate silent transitions and unreachable states, and compute spectral radii of adjacency matrices (their sizes are 38x38 and 40x40): $\#L_-^{0.05}(n) \sim 12.41^n$, $\#L_+^{0.05}(n) \sim 13.05^n$. Hence $12.41^n \cdot 0.05^n \leq V_n \leq 13.05^n \cdot 0.05^n$, and the entropy

$$\mathcal{H} \in [\log 0.62; \log 0.653] \subset (-0.69; -0.61).$$

Taking a smaller $\varepsilon = 0.01$ provides a better estimate for the entropy:

$$\mathcal{H} \in [\log 0.6334; \log 0.63981] \subset (-0.659; -0.644).$$

We proved in 4.4.1 that the true value of the entropy is $\mathcal{H} = \log(2/\pi) \approx \log 0.6366 \approx -0.6515$.

4.6 Kolmogorov Complexity of Timed Words

To interpret the results above in terms of information content of timed words we state, using similar techniques, some estimates of Kolmogorov complexity of timed words. Recall first the basic definition from [37] (see also [40]). Given a partial computable function (decoding method) $f : \{0;1\}^* \times B \to A$, a description of an element $x \in A$ knowing $y \in B$ is a word w such that $f(w, y) = x$. The Kolmogorov complexity of x knowing y, denoted $K_f(x|y)$ is the length of the shortest description. According to Kolmogorov-Solomonoff theorem, there exists the best (universal) decoding method providing shorter descriptions (up to an additive constant) than any other method. The complexity $K(x|y)$ with respect to this universal method represents the quantity of information in x knowing y.

Coming back to timed words and languages, remark that a timed word within a "simple" timed language can involve rational delays of a very high complexity, or even uncomputable real delays. For this reason, we consider timed words with finite precision ε. For a timed word w and $\varepsilon = 1/N$ we say that a timed word v is a rational ε-approximation of w if all delays in v are rational and $w \in \mathcal{B}_\varepsilon^{NE}(v)$[8].

Theorem 7. *Let $\mathcal{A}$ be a timed automaton satisfying A2-A4, L its language, $\mathcal{H}$ its entropy. For any rational $\alpha, \varepsilon > 0$, and any $n \in \mathbb{N}$ large enough there exists a timed word $w \in L$ of length n such that the Kolmogorov complexity of all the rational ε-approximations v of the word w is lower bounded as follows*

$$K(v|n, \varepsilon) \geq n(\mathcal{H} + \log 1/\varepsilon - \alpha). \tag{4.19}$$

Proof. By definition of the entropy, for n large enough

$$V_n > 2^{n(\mathcal{H}-\alpha)}.$$

[8] In this section, we use such South-West approximations v for technical simplicity only.

Consider the set S of all timed words v violating the lower bound (4.19)

$$S = \{v \mid K(v|n,\varepsilon) \leq n(\mathcal{H} + \log(1/\varepsilon) - \alpha)\}.$$

The cardinality of S can be bounded as follows:

$$\#S \leq 2^{n(\mathcal{H}+\log(1/\varepsilon)-\alpha)} = 2^{n(\mathcal{H}-\alpha)}/\varepsilon^n.$$

Applying Lemma 4 we obtain

$$\mathrm{Vol}(\mathcal{B}_\varepsilon^{NE}(S)) \leq \varepsilon^n \#S \leq 2^{n(\mathcal{H}-\alpha)} < V_n.$$

We deduce that the set L_n of timed words from L of length n cannot be included into $\mathcal{B}_\varepsilon^{NE}(S)$. Thus, there exists a word $w \in L_n \setminus \mathcal{B}_\varepsilon^{NE}(S)$. By construction, it cannot be approximated by any low-complexity word with precision ε. □

Theorem 8. *Let $\mathcal{A}$ be a timed automaton satisfying A2-A4, L its language, $\alpha > 0$ a rational number. Consider a "bloated" automaton $\mathcal{A}'$, which is like $\mathcal{A}$, but in all the guards each constraint $x \in [l,u]$ is replaced by $x \in [l \dot{-} \alpha, u+\alpha]$. Let $\mathcal{H}'$ be the entropy of its language. Then the following holds for any $\varepsilon = 1/N \in (0;\alpha/D)$, and any n large enough.*

> *For any timed word $w \in L$ of length n, there exists its ε-approximation v with Kolmogorov complexity upper bounded as follows:*
>
> $$K(v|n,\varepsilon) \leq n(\mathcal{H}' + \log 1/\varepsilon + \alpha).$$

Proof. Denote the language of $\mathcal{A}'$ by L', the set of words of length n in this language by L'_n and its n-volume by V'_n. We remark that for n large enough

$$V'_n < 2^{n(\mathcal{H}'+\alpha/2)}.$$

Let now $w = t_1 a_1 \dots t_n a_n$ in L_n. We construct its rational ε-approximation as in Lemma 5: $v = \underline{t_1} a_1 \dots \underline{t_n} a_n$. To find an upper bound for the complexity of v we notice that $v \in U$, where U is the set of all ε-timed words u of n letters such that $\mathcal{B}_\varepsilon^{NE}(u) \subset L'_n$. Applying Lemma 4 to the set U we obtain the bound

$$\#U \leq V'_n/\varepsilon^n < 2^{n(\mathcal{H}'+\alpha/2)}/\varepsilon^n.$$

Hence, in order to encode v (knowing n and ε) it suffices to give its number in a lexicographical order of U, and

$$K(v|n,\varepsilon) \leq \log \#U + c \leq n(\mathcal{H}' + \log 1/\varepsilon + \alpha/2) + c \leq n(\mathcal{H}' + \log 1/\varepsilon + \alpha)$$

for n large enough.

□

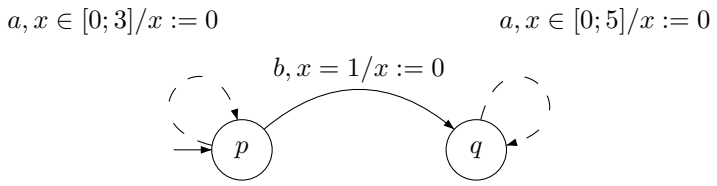


Figure 4.7: A pathological automaton

Two theorems above provide close upper and lower bounds for complexity of ε-approximations of elements of a timed language.

However, the following example shows that because we removed Assumption A5, in some cases these bounds do not match and $\mathcal{H}'$ can possibly not converge towards $\mathcal{H}$ when α becomes small.

Example 5. *Consider the automaton of Fig. 4.7. For this example, the state q does not contribute to the volume, and $\mathcal{H} = \log 3$. Nevertheless, when we bloat the guards, both states become "usable" and, for the bloated automaton $\mathcal{H}' \approx \log 5$. As for Kolmogorov complexity, for ε-approximations of words from the sublanguage $1b([0;5]a)^*$ it behaves as $n(\log 5 + \log(1/\varepsilon))$. Thus, for this bothering example, the complexity matches $\mathcal{H}'$ rather than $\mathcal{H}$.*

4.7 Conclusions and Further Work

In this chapter, we have defined size characteristics of timed languages: volume and entropy. The entropy has been characterized as logarithm of the leading eigenvalue of a positive operator on the space of continuous functions on a part of the state space. Three procedures have been suggested to compute it.

Research in this direction is very recent, and many questions need to be studied. We are planning to explore practical feasibility of the procedures described here and compare them to each other. We believe that, as usual for timed automata, they should be transposed from regions to zones. We will explore potential applications mentioned in the introduction.

Many theoretical questions still require exploration. Ongoing research is concerning with estimation of the gap between our upper and lower bounds for the entropy (we believe that this gap tends to 0 for strongly connected automata), with one goal being to establish entropy computability. We would be happy to remove some of Assumptions A1-A5. For instance we used A4 for ruling out some Zeno behaviors. As those truly Zeno behaviors do not actually contribute to the volume, this calls for a weakened assumption. Kolmogorov complexity estimates can also be improved, in particular, as shows Example 5, it could be more suitable to use another variant of entropy, perhaps $\mathcal{H}^+ = \max \mathcal{H}_q$, where the entropy is maximized with respect to initial states q. Extending results to probabilistic timed automata is another option. Our entropy represents the amount of

information per timed event. It would be interesting to find the amount of information per time unit. Another research direction is to associate a dynamical system (a subshift) to a timed language and to explore entropy of this dynamical system.

Chapter 5

Conclusion

Let us recapitulate the major contributions in the thesis:

- Scheduling: The model of streams of structured jobs is a rather realistic model for reactive scheduling. The observation made in this thesis about the conflict that may arise between backlog and latency can also be seen as a conflict between a kind of "fairness" (bounded latency for all) and the performance of the system as a whole. Our results made explicit what practitioners who have to pipeline activities (for example, in restaurants) probably know by doing. Some more studies on the pipelinability of sets of jobs may be interesting.

- Mean-Payoff Languages: Our main contribution to this domain is first by removing some ad-hoc features of the acceptance conditions found in literature, and then by finding a class of mean-payoff languages closed under Boolean operations and studying its properties. Our approach could certainly be adapted to other well known payoff criteria for similar expressivity, closure and analyzability results and then translated to games. It is easy to see there are many possible combinations to consider.

- Volume and Entropy: In this major part of the thesis we have adapted the quantitative measures of languages (size and entropy) to the dense-time domain. This work connects the theory of timed automata to the rich mathematics of linear operators. The definitions on the proof techniques developed in this part of the work may be useful to other applications, for example to analyze systems with probabilistic durations. The current work on our agenda are: to find conditions for which the we can prove that computation of the entropy is guaranteed to converge, to define a variant of volume and entropy which is indexed by the real time rather than by the number of events, to study these definitions as a basis for a theory of information for Boolean signals and to apply the technique to assess quality of timed language approximations.

We hope to have made some significant contributions to bridging the gap between qualitative and quantitative approaches to evaluate system behaviors.

Bibliography

[1] Karine Altisen, Gregor Gößler, Amir Pnueli, Joseph Sifakis, Stavros Tripakis, and Sergio Yovine. A framework for scheduler synthesis. In *IEEE Real-Time Systems Symposium*, pages 154–163, 1999.

[2] Rajeev Alur, Aldric Degorre, Oded Maler, and Gera Weiss. On omega-languages defined by mean-payoff conditions. In Luca de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2009.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *REX Workshop*, pages 74–106, 1991.

[5] Rajeev Alur, Aditya Kanade, and Gera Weiss. Ranking automata and games for prioritized requirements. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2008.

[6] Eugen Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.

[7] Eugene Asarin and Aldric Degorre. Volume and entropy of regular timed languages: Analytic approach. To appear in proceedings of FORMATS'09, 2009.

[8] Eugene Asarin and Aldric Degorre. Volume and entropy of regular timed languages: Discretization approach. To appear in proceedings of Concur'09, 2009.

[9] Eugene Asarin, Oded Maler, and Amir Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, pages 1–20, 1994.

[10] Eugene Asarin, Oded Maler, and Amir Pnueli. On discretization of delays in timed automata and digital circuits. In *CONCUR'98*, LNCS 1466, pages 470–484. Springer-Verlag, 1998.

[11] Eugene Asarin and Alexei Pokrovskii. Use of the Kolmogorov complexity in analyzing control system dynamics. *Automation and Remote Control*, (1):25–33, 1986.

[12] Ramzi Ben Salah, Marius Bozga, and Oded Maler. Compositional timing analysis. In *EMSOFT'09*. ACM, 2009.

[13] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Quantitative model-checking of one-clock timed automata under probabilistic semantics. In *QEST'08*, pages 55–64. IEEE Computer Society, 2008.

[14] Jacek Blazewicz, Klaus Ecker, Erwin Pesch, Günter Schmidth, and Jan Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer, 2nd edition, 2001.

[15] Lawrence Bodin, Bruce Golden, Arjang Assad, and Michael Ball. Routing and scheduling of vehicles and crews : The state of the art. *Computers & OR*, 10(2):63–211, 1983.

[16] Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queuing theory. *J. ACM*, 48(1):13–38, 2001.

[17] A.A. Brudno. Entropy and the complexity of the trajectories of a dynamical system. *Trans. Moscow Math. Soc.*, 44:127–151, 1983.

[18] Giacomo Bucci, Riccardo Piovosi, Luigi Sassoli, and Enrico Vicario. Introducing probability within state class analysis of dense-time-dependent systems. In *QEST'05*, pages 13–22. IEEE Computer Society, 2005.

[19] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Schedulding Algorithms and Applications*. Real-Time Systems Series. Springer, 2nd edition, 2005.

[20] Marco Caccamo, Theodore P. Baker, Alan Burns, Giorgio C. Buttazzo, and Lui Sha. Real-time scheduling for embedded systems. In D. Hristu-Varsakelis and W. Levine, editors, *Handbook of Networked and Embedded Control Systems*, pages 173–196. Birkhäuser, 2005.

[21] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *CONCUR*, pages 66–80, 2005.

[22] Krishnendu Chatterjee. Concurrent games with tail objectives. *Theor. Comput. Sci.*, 388(1-3):181–198, 2007.

[23] Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. The complexity of quantitative concurrent parity games. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 678–687, 2006.

[24] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. In *Proceedings of CSL 2008: Computer Science Logic*, Lecture Notes in Computer Science. Springer-Verlag, 2008.

[25] Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdziński. Mean-payoff parity games. In *Proceedings of the 20th Annual Symposium on Logic in Computer Science*, pages 178–187. IEEE Computer Society Press, 2005.

[26] Vicent Cholvi and Juan Echagüe. Stability of fifo networks under adversarial models: State of the art. *Computer Networks*, 51(15):4460–4474, 2007.

[27] Alain Darte, Yves Robert, and Frederic Vivien. *Scheduling and Automatic Parallelization*. Birkhauser Boston, 2000.

[28] Aldric Degorre and Oded Maler. On scheduling policies for streams of structured jobs. In Franck Cassez and Claude Jard, editors, *FORMATS*, volume 5215 of *Lecture Notes in Computer Science*, pages 141–154. Springer, 2008.

[29] Hesham El-Rewini. Partitioning and scheduling. In A. Zomaya, editor, *Parallel & Distributed Computed Handbook*, chapter 9, pages 239–273. McGraw-Hill, 1996.

[30] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theor. Comput. Sci.*, 354(2):301–317, 2006.

[31] Chai-Hien Gan, Phone Lin, Nei-Chiung Perng, Tei-Wei Kuo, and Ching-Chi Hsu. Scheduling for time-division based shared channel allocation for UMTS. *Wirel. Netw.*, 13(2):189–202, 2007.

[32] Hugo Gimbert and Wieslaw Zielonka. Deterministic priority mean-payoff games as limits of discounted games. In *ICALP*, pages 312–323, 2006.

[33] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In *ICALP'92*, LNCS 623, pages 545–558. Springer-Verlag, 1992.

[34] Guang-Hui Hsu. A survey of queueing theory. *Ann. Oper. Res.*, 24(1-4):29–43, 1990.

[35] Anant Singh Jain and Sheik Meeran. A state-of-the-art review of job-shop scheduling techniques, 1998.

[36] Edward G. Coffman Jr., editor. *Computer and Job-Shop Scheduling Theory*. J. Wiley, New York, 1976.

[37] Andreï N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1(1):1–7, 1965.

[38] M. A. Krasnosel'skij, E.A. Lifshits, and A. V. Sobolev. *Positive Linear Systems: The method of positive operators*. Number 5 in Sigma Series in Applied Mathematics. Heldermann Verlag, Berlin, 1989.

[39] Orna Kupferman and Yoad Lustig. Lattice automata. In *Proc. 8th Intl. Conf. Verification, Model Checking, and Abstract Interpretation*, LNCS 4349, pages 199–213. Springer, 2007.

[40] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 3 edition, 2008.

[41] Douglas Lind and Brian Marcus. *An introduction to symbolic dynamics and coding*. Cambridge University Press, 1995.

[42] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[43] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS*, pages 229–242, 1995.

[44] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer-Verlag, 1991.

[45] Dominique Perrin and Jean Éric Pin. *Infinite Words. Automata, Semigroups, Logic and Games*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

[46] Michael Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer Series in Operations Research and Financial Engineering. Springer, 2007.

[47] Amir Pnueli. The temporal logic of programs. In *18th IEEE Symposium on the Foundations of Computer Science (FOCS'77)*, pages 46–57, 1977.

[48] Cristobal Rojas. Computability and information in models of randomness and chaos. *Mathematical Structures in Computer Science*, 18(2):291–307, 2008.

[49] Luigi Sassoli and Enrico Vicario. Close form derivation of state-density functions over dbm domains in the analysis of non-Markovian models. In *QEST'07*, pages 59–68. IEEE Computer Society, 2007.

[50] Wolfganag Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 133–191. Elsevier Science Publishers, 1990.

[51] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

[52] H. Wong-Toi and D. Dill. Synthesizing processes and schedulers from temporal specifications. In *CAV*, pages 272–281, 1990.

[53] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.

Appendix A

Résumé en Français

A.1 Introduction

Les langages formels sont des séquences sur un ensemble discret de symboles appelé *alphabet*. On les spécifie souvent par des formules dans une certaine logique, ou bien par des expressions rationnelles ou des automates discrets de types variés. La théorie actuelle est principalement *qualitative*, dans le sens où ses objets sont des séquence sur un temps discret, non-métrique, dans le sens où l'acceptation d'une séquence sur un automate dépend du fait que l'on visite ou non un état accepteur, et enfin dans le sens où la comparaison de langages est plus souvent considérée en termes d'inclusion, plutôt qu'en termes de mesures quantitatives.

Cette thèse est une contribution à l'étude de ces aspects souvent négligés, et présente des résultats relatifs à trois classes de problèmes.

Nous proposons d'abord un modèle d'ordonnancement dynamique où une plateforme doit exécuter des séquences de requêtes définies par un langage temporisé. Nous y prouvons quelques résultats sur deux aspects quantitatifs des ordonnancements: latence et retard accumulé.

Ensuite, nous nous intéressons à l'utilisation du coût moyen d'une exécution infinie sur un automate pondéré comme manière de définir un langage de mots infinis. Nous établissons des résultats quand à l'expressivité d'un tel formalisme et son analysabilité.

Enfin, nous définissons des notions de volume et d'entropie pour les langages temporisés réguliers. Nous proposons plusieurs méthodes différentes pour calculer ou bien approximer ces quantités. Nous nous basons en particulier sur l'analyse fonctionnelle et une méthode de discrétisation. Nous établissons un lien avec la théorie de l'information, en reliant notre entropie à la complexité de Kolmogorov.

A.2 Ordonnancement de flux de jobs structurés

Nous étudions une classe de problèmes d'ordonnancement combinant des aspects structurels liés à des dépendances entre tâches et des aspects dynamiques liés au fait qu'un

flux de tâche non déterministe arrive en permanence pendant l'exécution de l'ordonnanceur. Pour cette classe de problèmes, nous développons une politique d'ordonnancement qui peut garantir une accumulation bornée de retard d'exécution pour l'ensemble des flux admissibles. Nous montrons cependant qu'aucune politique de la sorte ne peut garantir des latences bornées pour l'ensemble des flux, à moins qu'une certaine marge de liberté ne soit assurée.

A.2.1 Le problème d'ordonnancement récurrent

Quelques notations à propos des mots temporisés. Dans ce chapitre, les flux de requêtes sont dénotés par des mots temporisés, éventuellement infinis. Nous rappelons qu'un mot temporisé (infini) est une séquence (infinie) de de symboles pris soit dans un alphabet fini Σ, soit dans les réels positifs, symbolisant alors un délai de longueur égale à la valeur de ce nombre réel. Ainsi $\tilde{u} = 3a_1 2a_2 a_3 6$ représente le comportement où après 3 unités de temps se produit l'événement a_1, après lequel s'écoulent 2 unités de temps avant que a_2 ne se produise, et ainsi de suite.

Les données du problème. Les données du problème d'ordonnancement considéré sont d'un côté une description du système qui va exécuter les tâches (la *plate-forme d'exécution*), et de l'autre une description de la demande.

La *plate-forme d'exécution* est définie par un ensemble fini M de types de ressource (ou machines) et par une fonction $R : M \to \mathbb{N}$ associant à chacun de ces types un entier naturel représentant le nombre d'instances du type de ressource.

Ces ressources sont utilisables et réutilisables par des *tâches*, pour lesquelles on se donne aussi un ensemble fini T de types. Chaque tâche est associée à un seul type de ressource (fonction $\mu : T \to M$) et à une durée (fonction $d : T \to \mathbb{R}_+$) qui peut être vue comme le temps d'exécution au pire. Une tâche du type $a \in T$ va ainsi s'exécuter de façon indivisible, non préemptible, sur une instance de $\mu(a)$, et l'occuper de manière exclusive pendant toute la durée d'exécution $d(a)$.

Nous nous donnons aussi, dans un problème donné, un ensemble fini $\mathcal{J}$ de types de *job*, chaque type de job étant en fait un graphe acyclique défini sur une partie de T. Nous supposons que chaque type de job est défini sur un ensemble de types de tâche disjoint.

La demande est ainsi modélisée par un langage de mots temporisés finis ou non sur l'alphabet $\mathcal{J}$. Ce langage est appelé le *générateur de flux de requêtes*, et ses éléments, naturellement, les *flux de requêtes*.

Nous ne considérons pas les problème trivialement non ordonnançable, dans le sens où la quantité de travail demandée sur chaque type de ressource serait supérieure à ce que la plate-forme peut fournir. Ainsi, pour un flux de requêtes σ et un réel positif α, nous disons que σ est α-libre s'il existe $b \in \mathbb{R}$ tel que pour n'importe quel intervalle de dates $(t, t']$, la somme des durées des tâches demandées par σ sur cet intervalle pour un type de ressource donné m est inférieure à $\alpha(t' - t)R(m) + b$.

Le flux est dit *admissible* si cela est vrai pour $\alpha = 1$, sous-critique si c'est vrai pour $\alpha < 1$. Nous ne considérons dans la suite que des générateurs dont tous les flux sont admissibles.

Ordonnancements. Un ordonnancement associe une date de début d'exécution à une instance de tâche donnée. Formellement, c'est une fonction $s : T \times \mathbb{N} \to \mathbb{R}_+ \cup \{\infty\}$ où $s(a, i)$ doit être interprété, si c'est un réel fini, comme la date d'exécution de la i-ième instance de la tâche a (dans le flux de requêtes de l'exécution courante) et, si c'est ∞, comme le fait que cette instance ne sera jamais exécutée.

Cette définition assez large n'exclut pas les ordonnancements qui ne sont pas réalisables par la plate-forme, ou *valides*. Dans la suite, nous dirons qu'un ordonnancement est valide (par rapport à un flux σ) si :

- s n'exécute pas une instance de tâche avant qu'elle ne soit demandée dans σ (non proactivité),
- s n'exécute une instance de tâche que si toutes les tâches qui la précèdent dans son instance de job ont fini d'être exécutées,
- s ne fait pas exécuter plus de tâches en simultané sur un type de ressource donné qu'il n'y a d'instances de la ressource dans la plate-forme.

Nous voulons aussi mesurer la qualité d'un ordonnancement. Pour cela, nous avons retenu deux critères:

Le premier est le résidu, c'est à dire la somme des durées des tâches (a, i) demandées avant la date t par le flux de requêtes et qui n'ont pas encore été démarrées à t ($s(a, i) > t$). Nous nous intéressons en particulier au fait qu'un ordonnancement ait un résidu borné ou non.

La seconde mesure est la latence, c'est à dire le temps qui s'écoule entre la date de requête d'un job et la fin de l'exécution de toutes ses tâches dans cet ordonnancement. La latence d'un ordonnancement par rapport à un flux de requête est la borne supérieure des latences des instances de jobs du flux de requête. Elle peut être infinie si un job n'est jamais terminé, ou bien dans le cas où les latences des instances de jobs divergent vers l'infini.

Il est assez immédiat qu'un ordonnancement de latence finie a aussi un résidu borné. La réciproque est cependant fausse, car un ordonnancement qui n'exécuterait jamais une certaine instance de tâche pourrait cependant exécuter toutes les autres sous un délai borné.

A.2.2 Résultat négatif

Nous montrons un théorème établissant qu'un flux peut ne pas admettre d'ordonnancement de latence finie, bien que le flux soit admissible.

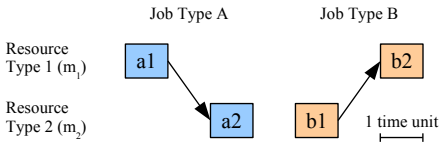


Figure A.1: Notre exemple.

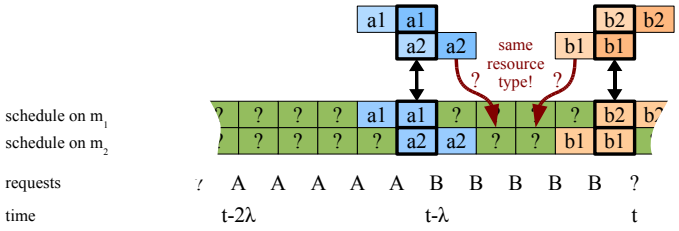


Figure A.2: Création d'un trou.

Pour cela, nous proposons un exemple de plate-forme d'exécution R sur $M = \{m_1, m_2\}$ avec $R(m_1) = R(m_2) = 1$ et l'ensemble de types de job $\mathcal{J} = \{A, B\}$, représenté sur la figure A.1, avec lesquels nous pouvons exhiber un tel flux, σ_∞.

Nous choisissons le flux σ_∞ de telle sorte qu'il inclue une infinité de séquences du type $A^{\lambda_i}B^{\lambda_i}$ pour des λ_i croissants. Nous montrons que pour un $l \in \mathbb{R}_+$ donné, le fait d'imposer une latence inférieure à l oblige à introduire un « trou » de longueur 1 dans l'ordonnancement à chaque facteur $A^\lambda B^\lambda$ tel que $\lambda > l$. Ici par « trou », nous désignons un intervalle de temps où une instance de ressource est inutilisée (Fig. A.2).

Une infinité de séquences $A^\lambda B^\lambda$ avec $\lambda > l$ dans σ_∞ implique ainsi une infinité de trous dans tout ordonnancement de latence inférieure à l, ce qui dans ce cas va rendre le résidu non borné, contredisant le fait que la latence est l.

Or ce raisonnement ne fait aucune hypothèse sur le réel positif l, ce qui veut dire que σ_∞ n'admet pas d'ordonnancement de latence finie.

A.2.3 Résultat positif

Politiques d'ordonnancement. Après nous êtres intéressés à l'ordonnancement d'un flux connu à l'avance, nous nous penchons sur l'ordonnancement sous incertitude : on sait que le flux qui sera observé sera l'un des éléments du générateur de flux de requêtes, mais on ignore lequel. Il s'agit ainsi de définir, hors-ligne, une *politique d'ordonnancement*, qui va, en ligne, générer des ordonnancements.

Formellement, une politique d'ordonnancement est une fonction associant à un préfixe fini de flux de requêtes un ensemble d'instance de tâches de ce préfixe, à interpréter comme l'ensemble des tâches à démarrer juste après avoir observé ce préfixe. Un autre point de vue est de considérer qu'une politique d'ordonnancement est un transducteur temporisé transformant un flux de requêtes en un ordonnancement.

Nous montrons dans cette partie qu'il existe une politique d'ordonnancement qui pour tout générateur admissible ne produit que des ordonnancements à résidu borné.

Premier arrivé, premier servi. Nous établissons d'abord qu'une politique naïve du type « premier arrivé, premier servi », ne peut pas assurer des résidus bornés pour n'importe quel flux admissible. L'argument de la preuve est le suivant : on montre que pour une telle politique d'ordonnancement, latence finie et résidu borné sont en fait des critères équivalents. Or nous avons déjà établi auparavant qu'il était impossible de garantir une latence finie pour certains flux. Pour un tel flux, une politique du type « premier arrivé, premier servi » n'aura donc pas un résidu borné.

Politique à résidu borné. L'échec de la politique naïve nous conduit donc à proposer une autre idée.

Le principe général de cette nouvelle proposition de politique est de faire en sorte qu'à tout moment, sous réserve que le résidu ait déjà une certaine taille, il existe au moins une instance de tâche prête à être exécutée pour chaque type de ressource, c'est-à-dire des instances dont les dépendances sont déjà satisfaites. Ainsi il va être possible, à n'importe quel instant, de démarrer une tâche sur toute ressource qui se libérerait.

Pour cela, nous créons une file FIFO pour chaque type de ressource, contenant des instances de tâches prêtes à être exécutées. Quand une ressource se libère, le premier élément de la file correspondante est exécuté s'il existe.

Les files, elles, se remplissent de la manière suivante : à chaque fois qu'une requête de job arrive, pour chacun des types de tâche constituant ce type de job, on choisit la plus ancienne instance prête, s'il en existe, et on la met en file.

Nous montrons qu'au pire des cas, quand la demande est égale à ce que la plate-forme peut traiter, il y a toujours une tâche en file quand une ressource se libère, ce qui implique que les files sont vidées aussi vite qu'elles se remplissent et qu'elles restent bornées.

Nous prouvons aussi, dans un résultat complémentaire, qu'une légère adaptation de cette politique permet d'assurer aussi une latence finie pour n'importe quel générateur sous-critique.

A.2.4 Discussion

Dans ce chapitre, nous avons donc prouvé quelques résultats fondamentaux sur un modèle qui, nous le croyons, rend compte de nombreux phénomènes de la vie réelle.

L'idée que des techniques issues de la vérification puissent être utilisées pour modéliser des problèmes difficiles à exprimer avec les modèles traditionnels de l'ordonnancement temps réel n'est pas nouvelle. En particulier, les algorithmes de vérification et de synthèse pour automates temporisés ont été utilisées à cet effet. Cependant, jusque là, ces approches avaient le défaut de ne pas pouvoir passer à l'échelle. La politique présentée dans ce chapitre n'a pas ce problème et s'adapte à tous les cas prévus par notre modèle, pour peu que les flux de requêtes soient admissibles.

Dans le futur, nous pensons qu'il serait intéressant de considérer différentes extensions de ce modèle, en particulier remplacer l'analyse « au pire » par des probabilités sur les durées et les flux. Enfin, nous aimerions caractériser la classe la plus générale d'ensembles de jobs dont tous les flux admissibles ont un ordonnancement à latence bornée.

A.3 Omega-langages définis par une condition de salaire moyen

En vérification quantitative, on associe un salaire (ou un coût) aux transitions des automates, qui sont utilisés pour associer un salaire moyen aux comportements infinis. Dans ce chapitre, nous proposons de définir des ω-langages par des critères sur ces salaires moyens. Des conditions telles que « le nombre de messages perdus est négligeable » ne sont pas ω-régulières, mais pourtant spécifiables dans notre modèle. Nous montrons que pour la fermeture par intersection, on a besoin de considérer des salaires multidimensionnels. Nous soutenons que les conditions d'acceptation d'un mot doivent considérer l'ensemble des points d'accumulation de la séquence des salaires moyens de ses préfixes, et nous donnons une caractérisation précise de tels ensembles. Nous proposons la classe des langages de salaire moyen à seuils multiples, utilisant comme condition d'acceptation le fait qu'une combinaison booléenne d'inégalités comparant la valeur maximale ou minimale des points d'accumulation sur une certaine coordonnée à une constante: le seuil. Pour cette classe de langages, nous étudions l'expressivité, les propriétés de clôture, l'analysabilité et la complexité de Borel.

A.3.1 Automates à salaires multiples, langages de salaire multiple moyen

Nous travaillons sur des automates à salaires multiples. Il s'agit d'automates à états finis déterministes munis d'une fonction de salaire associant à chaque transition un salaire dans $\mathbb{R}^d$, où d est un entier naturel associé à l'automate, que nous appellerons sa dimension.

Chaque *exécution* finie, c'est-à-dire chaque séquence finie de transitions successives autorisées dans l'automate, se voit associer un salaire moyen qui est la somme des salaires des transitions empruntées divisée par le nombre de transitions.

Pour une exécution infinie, il est tentant de définir le salaire moyen comme la limite des salaires moyens des préfixes de l'exécution. Or cette limite n'existe pas forcément. Dans la littérature, pour le cas $d = 1$, il est courant alors de considérer la limite inférieure ($\liminf$) de la suite des salaires moyens.

Nous considérons, cependant, qu'un tel choix est arbitraire : en effet, la limite supérieure aurait pu être choisie, choix qui n'est pas neutre si par la suite on veut comparer cette limite avec un seuil constant. De plus, avec $d > 1$ se pose la question de savoir sur quelle(s) coordonnée(s) appliquer ces limites inférieures ou supérieures.

Dans une première approche, nous choisissons donc de conserver plus d'information, en considérant qu'il est pertinent, à la place d'une valeur unique, d'étudier l'ensemble des valeurs d'adhérence de la suite des salaires moyens. Pour une exécution w sur un automate $\mathcal{A}$, nous notons cet ensemble $\mathrm{Acc}_{\mathcal{A}}(w)$[1].

Nous montrons, ce qui aura un rôle dans les résultats d'analysabilité, que $\mathrm{Acc}\,\mathcal{A}(w)$ est une partie fermée, bornée et connexe de $\mathbb{R}^d$, et que toute partie de $\mathbb{R}^d$ qui a ces propriétés est, pour une certaine exécution w, sur un certain automate $\mathcal{A}$ égale à $\mathrm{Acc}\,\mathcal{A}(w)$.[2]

Ainsi, nous ramenons l'acceptation d'une exécution w sur un automate $\mathcal{A}$ à une condition sur $\mathrm{Acc}_{\mathcal{A}}(w)$. Dans le cas le plus général, si F est un prédicat sur les parties de $\mathbb{R}^d$, nous disons que l'exécution w est *acceptée* sur $\mathcal{A}$ par le prédicat F si $F(\mathrm{Acc}_{\mathcal{A}}(w))$. De même, un mot infini est *accepté* si son unique exécution sur $\mathcal{A}$ est acceptée, et nous appelons langage d'un automate $\mathcal{A}$ par la condition F, l'ensemble $L(\mathcal{A}, F)$ des mots dont l'exécution est acceptée sur $\mathcal{A}$ par le prédicat F.

Enfin, si $d = 1$, nous appelons *condition de seuil* tout prédicat F tel que $F(S) \equiv \mathrm{extr}\, S \bowtie C$ où $\mathrm{extr} \in \{\inf, \sup\}$, $\bowtie \in \{<, >, \leq, \geq\}$ et $C \in \mathbb{R}$.

A.3.2 Expressivité.

Nous établissons ensuite une série de résultats d'expressivité. Il s'agira aussi bien de situer les langages à salaire moyen par rapport à des repères connus (langages rationnels et hiérarchie de Borel) que de comparer différentes sous-classes de langages à salaire moyen entre elles.

Comparaison avec les langages rationnels. Premièrement, nous montrons sur deux contre-exemples que la classe des langages à salaires multiples moyens est incomparable avec la classe des langages rationnels.

L'exemple d'un langage de salaire moyen qui ne soit pas rationnel est très classique : nous utilisons l'alphabet $\{a, b\}$ et exprimons par une condition de salaire moyen le fait qu'un mot ait au moins deux fois plus de b que de a. En utilisant un lemme de pompage adapté, nous montrons par l'absurde que ce langage ne peut pas être rationnel.

Réciproquement, le langage rationnel $L = (a^*b)^\omega$ ne peut pas être défini par une condition de salaire moyen, quelle qu'en soit la dimension, car aucune condition de ce type ne peut distinguer un mot qui a très peu de b à l'infini d'un mot qui n'en a plus du tout après un certain préfixe. Étant donné un automate à salaires multiples, on montre en effet qu'il y aura toujours un mot de L et un mot de son complément qui ont les mêmes valeurs d'adhérence sur cet automate.

[1] Acc pour « accumulation points », traduction trompeuse de « points d'adhérence » et non de « points d'accumulation »

[2] En fait, mieux que cela, pour un automate donné, pour peu que S soit une partie de l'enveloppe convexe des salaires d'une composante fortement connexe accessible de $\mathcal{A}$, on peut trouver une exécution w telle que $S = \mathrm{Acc}\,\mathcal{A}(w)$.

Comparaison de langages de salaire moyen à seuil. Nous discutons ensuite de l'expressivité comparée des classes de langages à salaire moyen que l'on peut définir à l'aide de simples seuils en dimension 1.

Pour $\bowtie \in \{<, >, \leq, \geq\}$, nous définissons la classe $\mathcal{L}_{\bowtie}$ des langages reconnaissables sur un automate à salaires unidimensionnels par une condition de seuil simple utilisant la relation $\bowtie$ et la borne inf. Bien qu'il s'agisse de variations autour de la condition habituellement utilisée dans la littérature sur les jeux de salaires moyens, les implications du choix de $\bowtie$ ne sont en général pas discutées, laissant penser que celui-ci est indifférent. Cependant, nous montrons ici que chacune des quatre possibilités aboutit à une classe de langages incomparable avec les trois autres.

La preuve est assez similaire à celle utilisée pour montrer que $(a^*b)^\omega$ n'est pas un langage à salaire moyen : on se donne un langage L bien choisi, appartenant à une classe $\mathcal{L}_{\bowtie_1}$, on se donne un automate muni d'une condition de seuil utilisant $\bowtie_2 \neq \bowtie$ B et on montre que le fait d'accepter certains mots de L sur cet automate oblige à accepter des mots du complément de L (ou parfois le contraire).

Langages à salaire moyen dans la hiérarchie de Borel. Nous situons ensuite les classes $\mathcal{L}_<$ et $\mathcal{L}_\leq$ dans la hiérarchie de Borel, établissant les quatre relations suivante : $\mathcal{L}_\leq \subset \Pi_2^0$, $\mathcal{L}_\leq \not\subseteq \Sigma_2^0$, $\mathcal{L}_< \subset \Sigma_3^0$ et $\mathcal{L}_< \not\subseteq \Pi_3^0$.

De l'utilité d'augmenter la dimension. Nous avons montré que le choix d'un opérateur de comparaison est en fait restrictif. Maintenant nous prouvons que, si nous sommes intéressés par des problèmes quantitatifs multicritères, le fait de se limiter à des conditions sur des salaires unidimensionnels est aussi restrictif.

Plus précisément, nous établissons que l'intersection d'un langage à salaires moyens défini en dimension d_1 et d'un langage de salaire moyen défini en dimension d_2 est toujours un langage de salaire moyen de dimension $d_1 + d_2$, et n'est en général pas un langage à salaire moyen de dimension $d < d_1 + d_2$.

Ce résultat est conforme à l'intuition que donnerait l'algèbre linéaire avec des considérations sur l'indépendance linéaire des salaires sur les différentes transitions, mais sa preuve exige cependant un raisonnement fin sur la structure en composantes fortement connexes et les comportements infinis de l'automate proposé pour reconnaître l'intersection.

A.3.3 Une classe analysable de langages à salaire moyen

Nous proposons une classe de langages à salaire moyen close par opérations booléennes, contenant les quatre classes $\mathcal{L}_<, \mathcal{L}_>, \mathcal{L}_\leq$ et $\mathcal{L}_\geq$ et analysable dans le sens où le problème du vide y est décidable.

À cette fin, nous considérons, comme conditions F, des combinaisons booléennes de conditions de seuil simples sur les projections le long des axes de l'ensemble des points d'accumulation d'une exécution. Et, comme classe analysable, nous choisissons

l'ensemble des langages reconnaissables sur un automate à salaires multiples par une telle condition F.

Nous démontrons d'abord que, comme on pouvait s'y attendre, cette classe est exactement la clôture de $\mathcal{L}_< \cup \mathcal{L}_> \cup \mathcal{L}_\leq \cup \mathcal{L}_\geq$ par opérations booléennes.

Enfin, nous donnons un algorithme permettant de décider si un langage défini défini par une telle condition sur un automate à salaires multiples est vide. Cet algorithme suppose que la condition d'acceptation est donnée sous forme normale disjonctive et cherche, pour chaque conjonction de la formule et pour chaque composante fortement connexe de l'automate, si une exécution finissant dans cette composante peut satisfaire la conjonction. C'est-à-dire que l'exécution doit avoir un ensemble de points d'accumulation satisfaisant tous les seuils apparaissant dans la conjonction.

A.3.4 Conclusion

Les seuils sur les salaires moyens étaient jusque là surtout utilisés en tant que condition de victoire dans certains jeux infinis. Dans ce chapitre, nous avons décidé d'utiliser plutôt ce type de condition pour définir des langages et établir une liste de résultats les concernant.

Que l'on souhaite soit prendre en compte de multiples critères quantitatifs, soit simplement bénéficier d'un formalisme de spécification clos par opérations booléennes, il ressort de notre étude qu'il est nécessaire de considérer des salaires multidimensionnels. Nous avons exhibé en particulier la plus petite classe de langages à salaires moyens multiples incluant les comparaisons à des seuils et ayant cette propriétés de clôture. Nous avons montré que le problème du vide y était décidable.

Après avoir obtenu ces résultats sur les langages, il serait logique, dans un travail futur, de se demander quelles seraient les implications de l'utilisation de salaires multiples dans les jeux de salaire moyen. Une autre voie à explorer serait de tenter de rétablir ces résultats dans le cas où l'automate serait non déterministe.

A.4 Volume et entropie des langages temporisés

Nous définissons des mesures de taille pour les langages temporisés : le volume pour un langage à nombre d'événements fixé, et l'entropie (vitesse de croissance) en tant que mesure asymptotique pour un nombre d'événements non borné. Ces mesures peuvent être utilisées pour une comparaison quantitative de langages et l'entropie peut être vue comme la quantité d'information d'un langage temporisé. Pour les langages acceptés par les automates déterministes, nous donnons des formules exactes pour le volume. Ensuite nous caractérisons l'entropie en utilisant des méthodes d'analyse fonctionnelle, en tant que logarithme de la valeur propre principale (ou rayon spectral) d'un opérateur intégral positif.

Nous établissons plusieurs méthodes pour calculer l'entropie : une symbolique pour les automates que nous appelons à « une horloge et demie », et deux numériques :

une qui utilise encore des techniques d'analyse fonctionnelle, et l'autre qui est basée sur la discrétisation. Nous donnons une interprétation de notre entropie en théorie de l'information, en termes de complexité de Kolmogorov.

A.4.1 Exposé du problème

Volume et entropie des langages temporisés. Pour un mot temporisé à n événements discrets $w = t_1 a_1 t_2 \dots t_n a_n$, nous définissons deux notions : son *timing*, c'est-à-dire $\theta(w) = (t_1, \dots, t_n)$ in $\mathbb{R}^n$ et son *untiming* $\eta(w) = a_1, \dots, a_n \in \Sigma^n$ (un mot non temporisé).

Nous nous servons de ces notions pour définir le *n-volume* V_n d'un langage temporisé L:

$$V_n(L) = \sum_{v \in \Sigma^n} \mathrm{Vol}\{\theta(w) \mid w \in L, \eta(w) = v\},$$

Il s'agit donc de la somme des volumes euclidiens en dimension n des parties de $\mathbb{R}^n$ décrites par les timings possibles d'un même untiming (notion qui est bien définie pour les langages réguliers, vu qu'il s'agit de polyhèdres).

Quand n tend vers l'infini, ce volume très souvent diverge exponentiellement ou, au contraire, tend vers zéro. C'est pour cela que nous nous intéressons plus particulièrement à sa (dé)croissance asymptotique et que nous définissons l'*entropie* d'un langage comme suit: $\mathcal{H}(L) = \limsup_{n\to\infty} \frac{\log V_n}{n}$.

Trois exemples. Nous donnons ensuite quelques exemples d'automates. Deux dont les volumes successifs et l'entropie de leurs langages peuvent être calculés explicitement sans technique particulière, et un troisième, notre exemple favori, qui résiste à une analyse naïve, et pour lequel nous arrivons seulement à calculer le volume du langage, pour un n donné, comme une série de n intégrales imbriquées, ce qui ne nous permet pas de déduire son entropie. C'est sur cet exemple que nous illustrons les techniques proposées dans la suite du chapitre.

Une sous-classe d'automates temporisés. Nos méthodes s'appliquent à des automates temporisés satisfaisant quelques hypothèses relativement peu restrictives. Ainsi, dans la suite, nous utilisons toutes ou une partie des hypothèses suivantes :

A1. L'automate $\mathcal{A}$ est déterministe

A2. Tous ses états sont accepteurs (on s'intéresse seulement aux langages préfixe-clos)

A3. Les gardes sont rectangulaires, c'est à dire sont des conjonctions de contraintes $L_i \le x_i \le U_i$. Au moins une horloge doit être bornée.

A4. Il existe une constante $D \in \mathbb{N}$ telle que tout segment d'exécution de D transitions réinitialise toutes les horloges.

A5. Aucune garde n'est ponctuelle, c'est-à-dire dans toute garde, pour tout i, $L_i < U_i$.

La plupart de nos résultats nécessitent de mettre l'automate sous forme dite *séparée par régions*, c'est-à-dire sous la forme d'un automate $\mathcal{A} = (Q, \Sigma, C, \delta, q_0)$, ayant les propriétés suivantes, en plus de A1, A2 et A4:

B1. Chaque état discret et chaque transition est atteignable depuis $(q_0, 0)$

B2. Pour chaque état discret $q \in Q$, il existe une unique région d'horloges $\mathbf{r}_q$ telle que l'ensemble des valeurs d'horloges par lequel q est entré est exactement $\mathbf{r}_q$. On pose $\mathbf{r}_{q_0} = \{0\}$.

B3. Pour toute transition discrète, la garde est aussi une région d'horloges.

Nous montrons qu'il est possible de construire, à partir d'un automate satisfaisant $A1 - A4$, un automate séparé par régions reconnaissant le même langage.

Calcul du volume. Nous exprimons le volume du langage reconnu par un automate satisfaisant $A1 - A3$ grâce à une formule récurrente, déduite des équations récurrente du langage de l'automate.

Nous notons L_n les mots à n événements acceptés par $\mathcal{A}$.

Alors L_n vérifie les équations récurrentes de langage suivantes (avec $L_n = L_n(q_0, 0)$) :

$$\begin{aligned} L_0(q, \mathbf{x}) &= \varepsilon; \\ L_{k+1}(q, \mathbf{x}) &= \bigcup_{(q,a,\mathfrak{g},\mathfrak{r},q') \in \Delta} \bigcup_{\tau : \mathbf{x}+\tau \in \mathfrak{g}} \tau a L_k(q', \mathfrak{r}(\mathbf{x}+\tau)), \end{aligned}$$

où $L_n(q, \mathbf{x})$ est le langage des exécutions à n événements depuis l'état temporisé $(q, \mathbf{x})$.

Ainsi, en posant $v_n(q, x) = V_n(L_n(q, \mathbf{x}))$, nous établissons des équations similaires pour le volume :

$$v_0(q, \mathbf{x}) = 1; \tag{A.1}$$

$$v_{k+1}(q, \mathbf{x}) = \sum_{(q,a,\mathfrak{g},\mathfrak{r},q') \in \Delta} \int_{\tau : \mathbf{x}+\tau \in \mathfrak{g}} v_k(q', \mathfrak{r}(\mathbf{x}+\tau))\, d\tau. \tag{A.2}$$

De là nous déduisons que $v_n(q, \mathbf{x})$ est un polynôme de degré n à coefficients rationnels que l'on sait calculer, et que donc le n-volume du langage de $\mathcal{A}$ est un nombre rationnel calculable à partir de $\mathcal{A}$.

A.4.2 Approche par opérateurs

Définition de l'opérateur et lien avec l'entropie. Nous faisons le constat que la récurrence sur les volumes peut s'écrire sous la forme $v_{k+1} = \Psi v_k$, où Ψ est un opérateur linéaire intégral positif sur les fonctions réelles continues sur l'espace des états. Ceci

nous permet d'écrire la fonction v_n sous la forme fermée $v_n = \Psi^n 1$ et ramène le problème calcul du volume et de l'entropie à l'étude des itérations de cet opérateur.

Nous montrons en particulier que l'opérateur Ψ est compact, et en nous inspirant de la théorie générale des opérateurs linéaires positifs, nous déduisons notre théorème principal, à savoir que l'opérateur Ψ associé à un automate $\mathcal{A}$ satisfaisant $A1$, $A2$ et $A4$ a un rayon spectral dont le logarithme coïncide avec l'entropie du langage de $\mathcal{A}$.

Calcul de l'entropie pour automates à « une horloge et demie ». Nous isolons une classe d'automates, dits *à une horloge et demie*, pour laquelle le rayon spectral de l'opérateur, et donc l'entropie, peut être calculée symboliquement en résolvant une équation transcendante.

Un automate à une horloge et demie est un automate séparé par régions, dont les régions d'entrée ont toutes une dimension 0 ou 1. Autrement dit, toute transition discrète laisse au plus une horloge non réinitialisée.

Pour cette classe d'automates, la fonction $v_n(q, \mathbf{x})$, à q fixé, ne dépend que d'une seule variable réelle, ce qui permet, après quelques étapes que nous expliquons en détail, de transformer le système intégral $\lambda f = \Psi f$ en un système différentiel linéaire ordinaire que nous savons résoudre. Le rayon spectral de Ψ est ainsi la plus grande valeur de λ telle que ce système ait des solutions non nulles, condition qui dans nos calculs se ramène à chercher le plus grand λ qui annule un déterminant dont les coefficients sont des polynômes d'exponentielles paramétrées par λ.

Nous appliquons la méthode à notre exemple favori, et nous trouvons une entropie de $\log \frac{2}{\pi}$.

Approximation du rayon spectral dans le cas général. Dans le cas général, nous montrons que s'il existe deux réels α et β et un entier naturel m tels que $\alpha v_m \leq v_{m+1} \leq \beta v_m$ et $V_m > 0$, alors $\log \alpha \leq \mathcal{H} \leq \log \beta$.

Cette inégalité donne lieu à un algorithme permettant d'encadrer l'entropie par itérations successives de l'opérateur. Nous n'avons pas montré la convergence de cet algorithme, cependant, dans le cas où l'automate est fortement connexe, celle-ci se vérifie expérimentalement.

A.4.3 Approche par discrétisation

Nous proposons ensuite une autre approche permettant de calculer une approximation de l'entropie, celle-ci basée sur la discrétisation de l'automate temporisé étudié.

Pour $\varepsilon > 0$, nous disons qu'un mot est ε-temporisé si tous ses délais sont multiples d'ε.

Nous montrons que le volume de L_n est à peu près égal au nombre de mots ε-temporisés de L_n fois ε^n.

Notre technique de calcul de l'entropie par discrétisation consiste ainsi à déduire de $\mathcal{A}$ un automate à compteurs sur l'alphabet $\Sigma \cup \{\tau\}$ dont le langage est, à un morphisme

près qui transforme les τ en des délais de valeur ε, l'ensemble des mots ε-temporisés de L.

Après avoir transformé l'automate à compteurs en automate fini à multiplicités, nous nous inspirons des techniques de Lind et Marcus pour en compter les mots à n événements et même obtenir la vitesse de croissance logarithmique de ce nombre. Il s'agit en fait du logarithme du rayon spectral ρ de la matrice d'adjacence de l'automate.

De l'automate temporisé, nous dérivons en réalité deux automates à compteurs dont les rayons spectraux obtenus par cette méthode sont ρ_- et ρ_+, tels que $\log \varepsilon\rho_- \leq L(\mathcal{A}) \leq \log \varepsilon\rho_+$.

Cette méthode produit un encadrement de l'entropie qui, empiriquement, pour les automates dont le graphe des transitions non ponctuelles est fortement connexe, converge quand epsilon tend vers zéro.

A.4.4 Complexité de Kolmogorov des mots temporisés

Nous avons défini une notion d'entropie que nous savons dans certains cas calculer et, dans tous les cas, approximer. Nous montrons maintenant que cette notion est pertinente, dans le sens où nous arrivons à la relier à la notion de complexité de Kolmogorov.

La complexité de Kolmogorov est le nombre minimal de symboles nécessaires pour définir un objet donné. Dans le cas d'un langage temporisé, la complexité d'un mot peut être aussi bien très faible dans le cas où le mot est obtenu, par exemple, en empruntant toujours les transitions aussitôt que sa garde est vraie, ou bien infinie si les transition sont empruntées après avoir attendu un délai réel non calculable.

Ainsi, afin d'établir une relation pertinente, nous avons décidé de considérer, pour un mot donné, le mot de complexité la plus faible sur un voisinage de rayon ε, et de trouver le maximum de ce minimum sur le langage des mots à n événements de l'automate temporisé $\mathcal{A}$. Nous notons cette quantité $K(L_n, \varepsilon)$.

Nous montrons pour tout $\alpha > 0$ et pour n assez grand que $n(\mathcal{H} + \log 1/\varepsilon - \alpha) \leq K(L_n, \varepsilon) \leq n(\mathcal{H}' + \log 1/\varepsilon + \alpha)$, où $\mathcal{H}'$ est l'entropie du langage de $\mathcal{A}'$, automate identique à $\mathcal{A}$, mais dont les gardes ont été élargies de α. Autrement dit, pour n grand, $K(L_n, \varepsilon)$ est à peu près égal à $n(\mathcal{H}' + \log 1/\varepsilon)$, ce qui veut dire que notre notion d'entropie peut être reliée à la quantité d'information contenue dans le langage observé sous précision ε.

A.4.5 Discussion

Nous avons dans ce chapitre défini les notions de volume et d'entropie pour les langages temporisés réguliers. Nous avons caractérisé l'entropie comme le logarithme du rayon spectral d'un certain opérateur linéaire positif sur un espace de fonctions continues. Trois procédures pour le calculer ou l'approcher ont été suggérées.

Pourtant, de nombreuses questions restent à étudier. En particulier sur la possibilité d'obtenir des algorithmes pratiques et de les comparer les uns aux autres. Il serait aussi intéressant d'explorer les applications que nous avons envisagées.

D'un point de vue théorique, nous souhaiterions supprimer ou alléger certaines des hypothèses. Nous aimerions aussi pouvoir raisonner sur les composantes fortement connexes de l'automate pour améliorer les estimations de l'entropie et prouver leur convergence. Nous avons par ailleurs commencé à étudier le calcul de l'entropie par unité de temps, au lieu de l'entropie par événement. Enfin nous aimerions faire le lien avec l'entropie des systèmes dynamiques en associant un subshift au langage temporisé étudié.

A.5 Conclusion

Pour récapituler, nous avons exploré trois domaines assez différents où des aspects quantitatifs des langages formles interviennent. Dans chacune de ces directions, nous espérons avoir réalisé des contributions significatives :

- des résultats fondamentaux concernant les performances dans un modèle d'ordonnancement que nous pensons capable de modéliser de nombreux phénomène réels,
- une manière de combiner plusieurs critères quantitatifs pour définir des ω-langages, avec les résultats d'expressivité, de clôture et d'analysabilité qui correspondent,
- et enfin, pour les langages temporisés nous avons défini des notions de volume et d'entropie que nous pouvons calculer ou approcher.